Tampereen yliopisto

Valtteri Forsman

# AUTOMAATIO-OHJELMISTOJEN KEHITTÄMINEN JA VERSIONHALLINTA

## Tapaustutkimus SRF-tuotantoprosessista

Teknisten ja luonnontieteiden tiedekunta
Diplomityö
Marraskuu 2019

# TIIVISTELMÄ

Valtteri Forsman: Automaatio-ohjelmistojen kehittäminen ja versionhallinta
Diplomityö
Tampereen yliopisto
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Marraskuu 2019

---

Automaatio-ohjelmistojen kompleksisuus on muodostunut ongelmaksi muun muassa prosessiteollisuudessa ja koneenvalmistuksessa. Ohjelmistojen kompleksisuuden ja kustannusten kasvaessa ohjelmistojen kehitykseen ja ylläpitoon tulee kiinnittää entistä enemmän huomiota. Ohjelmistojen kehitystä ja ylläpitoa tukevia menetelmiä ovat esimerkiksi konfiguraation- ja versionhallinta sekä mallipohjainen ja modulaarinen suunnittelu. Nämä systemaattisen ohjelmistokehityksen menetelmät eivät kuitenkaan ole vakiintuneita toimintatapoja pienissä ja keskisuurissa yrityksissä

Tutkimuksen tavoitteena on selvittää, miten edellä mainitut menetelmät tukevat ohjelmistokehitystä ja liiketoimintaa, erityisesti keskisuurissa teollisuusyrityksissä. Tutkimusstrategiaksi on valittu yksittäinen tapaustutkimus. Tutkimuksen kohteena olevalle tuotantolinjalle etsitään kuvailutapa, jossa osasysteemien väliset rajapinnat kuvataan materiaalin ja informaation virtauksena. Rajapintasopimukset mahdollistavat tuotantolinjan modulaarisen rakenteen. Tällöin muutokset yhdessä osasysteemissä eivät vaikuta muihin osasysteemeihin, jos rajapinnat säilyvät ennallaan. Kuvailutavan käyttötarkoituksena on tukea simulaattorikehitystä ja konfiguraationhallintaa. Simulaatioita voidaan hyödyntää ohjelmistojen kehittämisessä ja testaamisessa.

Versionhallinnalle määritetään tutkimuksessa selkeät tavoitteet ja vaatimukset sekä tutkitun yrityksen tarpeisiin soveltuva työkalu. Tutkimuksessa esitetään lähestymistapa, jossa ohjelmistojen ja simulointimallien versionhallinta integroidaan osaksi tuotantolinjalle määriteltyä tuotetietorakennetta. Tuotantolinjan tuotetieto pyritään kuvailemaan kokonaisvaltaisesti integroidulla tuotetietomallilla. Tuotetiedon kuvailu sisältää sekä tuotantolinjan ohjelmiston että siihen liittyvän laitteiston.

Tällä hetkellä yrityksen tuottamat automaatio-ohjelmistot ovat kuitenkin ei-modulaarisia. Tutkimuksessa esitetään lähestymistapa ohjelmistojen modulaarisen rakenteen määrittelemiseksi ja ohjelmistojen modulaarisointiin liittyvän siirtymästrategian luomiseksi. Ohjelmistomoduulien kehittämiseen vaadittavan investoinnin kannattavuutta arvioidaan takaisinmaksuaikaan perustuvalla menetelmällä. Lisäksi tuodaan esille investoinnin kustannusten ja takaisinmaksuajan arviointiin liittyviä epävarmuustekijöitä.

Modulaaristen ohjelmistojen kehittämisen tueksi ehdotetaan myös mallipohjaisen suunnitteluprosessin käyttöönottoa. Tutkimuksessa tuodaan esiin mallipohjaisen suunnittelun hyödyntämiseen liittyviä etuja ja haasteita. Tunnistettuja etuja ovat kehitettävien ohjelmistojen jatkuva testaaminen, toimintojen dokumentoinnin helppous ja mallien hyödyntäminen sidosryhmien (kuten alihankkijat) välisessä kommunikoinnissa. Mallipohjaisten menetelmien hyödyntämisen haasteina taas ovat standardoidun kehitysalustan puuttuminen sekä laite- ja laitosvarianttien suuri määrä. Lisäksi uusien ohjelmistokehitysmenetelmien käyttöönottaminen vaatii tietotaidon ja resurssien lisäämistä organisaation sisällä.

Avainsanat: ohjelmistokehitys, versionhallinta, modulaarisuus, mallipohjainen suunnittelu.

# ABSTRACT

Valtteri Forsman: Development and version control of automation software
Master of Science thesis
Tampere University
Master's degree programme in Automation Engineering
November 2019

---

The complexity of automation software has become a real problem in the process and machine industries. As the complexity and cost of software increase, more attention must be paid to the development and maintenance of software. Methods supporting the development and maintenance of software are configuration management and version control as well as model-based and modular design. However, the systematic software development methods in question have not been widely adapted by small and medium sized enterprises.

The objective of this study is to gain more information about utilization of the above-mentioned methods, especially in the medium sized industrial enterprises. An individual case study has been chosen as a research strategy. The subject of the case study is a single production line.

Functional decomposition method is used to analyse the production line. Interfaces between the subsystems of the production line are described as material and information flows. The interface agreements are used to facilitate the modular structure of the production line. Modularity principle means that changes in one subsystem will not affect to other subsystems if the interfaces remain unchanged. Modular structure of the production line supports simulator development and configuration management. The simulations can be utilized in the development and testing of software.

Objectives and requirements as well as suitable tools for the version control are determined in this study. Scope of the version control is limited to software and simulation models related to the production line. Approach used for the version control is to describe the production line with an integrated product data model. The product data consists of software and hardware of the production line.

In the current situation, the automation software implementations produced by the studied company are non-modular. An approach to define the modular structure of software is represented in this study as well as creation of the transition strategy related to the modularization of software. The transition process requires investment for the development of software modules. A method based on the payback period is used to evaluate the profitability of the investment. Furthermore, factors of uncertainty related to the cost and payback period of the investment are presented in this study.

Utilization of model-based design process also is proposed to support the development of modular software. Advantages and challenges related to the model-based design methods are presented in the study. The identified advantages are continuous testing of the software, reducing the effort related to documentation of functionalities and improvements in communication between the interest groups (such as the subcontractors) by utilizing the models. The challenges are the lack of the standardized development platform as well as the large number of machine variants and power plant variants. Furthermore, introduction of new software development methodologies requires increased level of know-how and resources inside the organisation.

Keywords: software development, version control, modularity, model-based design

# ALKUSANAT

Tämä diplomityöprojekti oli innostava ja opettavainen kokemus. Laajan tutkimuksen tekeminen oli kuin suuri seikkailu, jonka määränpää tuntui alussa vain kaukaiselta pisteeltä horisontissa. Tutkimusaihe oli monipuolinen ja erittäin mielenkiintoinen kokonaisuus, joka mahdollisti sekä olemassa olevan ammattitaitoni hyödyntämisen että ammatillisen kasvun ja kehityksen. Kaiken kaikkiaan diplomityöprosessiin mahtui paljon haasteita ja onnistumisen elämyksiä.

Diplomityön loppuunsaattaminen päättää myös akateemisen urani, ainakin toistaiseksi. Matka fuksista kohti diplomi-insinöörin tutkintoa on ollut ikimuistoinen, mistä kiitos kuuluu kaikille kanssaopiskelijoille ja kollegoille yliopistossa. Matkalle mahtuu paljon hyviä muistoja akateemisesta maalimasta niin kotimaassa kuin ulkomaillakin.

Erityisesti haluan kiittää professori Risto Ritalaa ja Assistant Professor David Hästbackaa asiantuntevasta ohjauksesta tutkimusprojektin aikana. Suuren kiitoksen ansaitsevat myös BMH:n ohjausryhmän jäsenet Ville Hakanperä, Mika Tuomola, Jussi Rautiainen, Toni Pirttilä ja Ari Huhta. Heidän tarjoamat näkökulmat olivat erittäin tärkeitä tutkimusongelmien hahmottamisen ja ratkomisen kannalta. Kiitos kuuluu myös Mikko Tiutulle ja Jouni Lepistölle hyvien neuvojen ja näkemyksien jakamisesta. Lisäksi haluan esittää nöyrimmän kiitoksen sekä läheisilleni että ystävilleni korvaamattomasta tuesta ja kannustuksesta diplomityöprojektin edetessä.

*Seinäjoella, 5.11.2019*

*Valtteri Forsman*

# SISÄLLYSLUETTELO

# KUVALUETTELO

# LYHENTEET JA MERKINNÄT

| | |
|---|---|
| 3D-malli | engl. three dimensional, kolmiulotteinen tietokonemalli |
| ALM | engl. Application Lifecycle Management, sovelluksen elinkaaren hallinta |
| AutomationML | engl. Automation Markup Language, tietoformaatti työkalujen väliseen tiedonvaihtoon |
| CAD | engl. Computer Aided Design, tietokoneavusteinen suunnittelu |
| CAEX | engl. Computer Aided Engineering Exchange, tietoformaatti hierarkkisen luokkarakenteen tallentamiseen |
| CSV-tiedosto | engl. Comma Separated Values, tekstitiedostomuoto taulukkorakenteen tallentamiseen |
| FAT | engl. Factory Acceptance Test, tehdashyväksyntätesti |
| FE-metalli | ferromagneettinen metalli |
| FMU | engl. Functional Mockup Unit, työkaluriippumaton rajapintastandardi simulointiin |
| HIL | engl. Hardware In the Loop, reaaliaikasimuloinnin tekniikka |
| I/O | engl. Input and Output, tulo ja lähtö |
| IEC | engl. International Electrotechnical Commission, kansainvälinen sähköalan standardointiorganisaatio |
| IEEE | engl. Institute of Electrical and Electronics Engineers, kansainvälinen tekniikan alan järjestö |
| ISO | engl. International Organization for Standardization, kansainvälinen standardisoimisjärjestö |
| KKS | saksaksi Kraftwerk Kennzeichen System, voimalaitoksen tunnistejärjestelmä |
| MAT-tiedosto | MATLABin käyttämä tiedostomuoto matriisien tallentamiseen |
| MDA | engl. Model Driven Architecture, mallikeskeinen arkkitehtuuri |
| MIPS | engl. Massive Impulse Protection System, murskainta suojaava patentoitu ominaisuus |
| PDM | engl. Product Data Management, tuotetiedon hallinta |
| PID | engl. Proportional Integral Derivative, PID-säädin |
| PL-luokitus | engl. Performance Level, turvallisuuteen liittyvä suoritustaso |
| PLC | engl. Programmable Logic Controller, ohjelmoitava logiikka |
| PLM | engl. Product Lifecycle Management, tuotteen elinkaaren hallinta |
| SAT | engl. Site Acceptance Test, laitoshyväksyntätesti |
| SI-järjestelmä | ransk. Système International d'unités, kansainvälinen mittayksikköjärjestelmä |
| SIL-luokitus | engl. Safety Integrity Level, turvallisuuden eheyden taso |
| SRF | engl. Solid Recovered Fuel, kiinteä kierrätyspolttoaine |
| STL-tiedosto | engl. stereolithography, mallin geometriaa kuvaava tiedostomuoto |
| SysML | engl. Systems Modeling Language, mallinnuskieli systeemisuunnitteluun |
| TSNC | engl. Tagname Signal Naming Convention, signaalien nimeämiskäytäntö |
| UML | engl. Unified Modeling Language, mallinnuskieli ohjelmistosuunnitteluun |
| XML | engl. Extensible Markup Language, laajennettava merkintäkieli |

| | |
|---|---|
| $m$ | massa |
| $\dot{m}$ | massan aikaderivaatta. |

# 1. JOHDANTO

## 1.1 Motivointia

Logiikkaohjaimien ja mikrokontrollerien suorituskyky on kasvanut huomattavasti viime vuosina. Edistyneissä laitteissa, yhä suurempi osuus toiminnallisuuksista toteutetaan ohjelmistoilla. Asiakaskohtainen räätälöinti on kallista, joten laitevalmistajat pyrkivät alustan standardointiin. Ohjelmistokehityksen osuus tuotekehityskustannuksista siis kasvaa suhteessa laitteistokehitykseen. Edistyneet ohjelmistoratkaisut ovat valmistajille myös keino erottua edukseen markkinoilla. Ohjelmistojen räätälöintiin liittyvät kustannukset saattavatkin kasvaa moderneissa laitteissa huomattavan suuriksi.

Automaatio-ohjelmistojen kompleksisuus on muodostunut ongelmaksi esimerkiksi prosessiteollisuudessa ja koneenvalmistuksessa. Ohjelmistojen kompleksisuuden ja kustannusten kasvaessa, ohjelmistojen kehitykseen ja ylläpitoon tulee kiinnittää entistä enemmän huomiota. Tuotettujen ohjelmistojen kehitystä ja ylläpitoa tukevia menetelmiä ovat esimerkiksi konfiguraation- ja versionhallinta sekä mallipohjainen ja modulaarinen suunnittelu. Edellä mainitut systemaattisen ohjelmistokehityksen menetelmät eivät kuitenkaan ole vakiintuneita toimintatapoja pienissä ja keskisuurissa yrityksissä.

Automaatio-ohjelmistojen kehitykseen ja versionhallintaan liittyvistä menetelmistä löytyy vain vähän aiempaa akateemista tai teollista tutkimusta. Aiempi tutkimus aiheen ympärillä on keskittynyt perinteisiin ohjelmistoprojekteihin. Reaaliaikajärjestelmiä ohjaavilla automaatio-ohjelmistoilla on kuitenkin omat erityispiirteensä, jotka koskevat erityisesti ohjelmistojen käytettävyysastetta, luotettavuutta ja turvallisuutta. Aiheesta on siis vain vähän empiiristä tietoa, joten tapaustutkimuksen soveltaminen automaatio-ohjelmistojen kehittämiseen ja versionhallintaan on hyvin perusteltua.

## 1.2 Tutkimusongelma

Tämän tutkimuksen pääongelmaa kuvaava tutkimuskysymys on ”Miten ohjelmistokehitystä voidaan edistää tutkitussa organisaatiossa?”. Pääongelman analysoimiseksi tutkimusongelma on jaettu seuraaviin alakysymyksiin:

- Millainen tuotantolinjan kuvailutapa tukee ohjelmistokehitystä?
- Miten systeemin konfiguraationhallinta tukee automaatio-ohjelmistojen versionhallintaa?
- Mitä työkaluja ohjelmistokehitykseen ja ohjelmistojen versionhallintaan tarvitaan?

- Miten mallipohjaista suunnittelua voidaan hyödyntää ohjelmistokehityksessä?

Tutkimusongelmaan vastataan kuvaamalla ohjelmistokehityksen nykytilannetta organisaatiossa ja soveltamalla kirjallisuuskatsauksessa esiteltäviä ratkaisumalleja tapaustutkimukseen sekä laatimalla toimenpide-ehdotus ohjelmistokehityksen edistämiseksi. Tutkittavaksi tapaukseksi on valittu yksittäinen kierrätyspolttoaineen tuotantolinja. Tapauksen kontekstina on koneenvalmistuksen ja prosessiteollisuuden alalla toimiva keskisuuri yritys. Tutkimuksen kontribuutiona on tuottaa lisää tietoa modernien kehitysprosessien ja toimintatapojen soveltamiseen liittyvistä mahdollisuuksista ja rajoitteista, erityisesti keskisuurissa teollisuusyrityksissä.

## 1.3 Tutkimuksen tavoitteet ja rajaukset

Tutkimuksen tavoitteena on laatia toimenpide-ehdotus, joka mahdollistaa vaiheittaisen siirtymisen systemaattiseen ohjelmistokehitykseen. Toimenpide-ehdotuksessa arvioidaan myös siirtymän taloudellista kannattavuutta, mikä edistää strategisen muutoksen läpiviemistä tutkitussa organisaatiossa. Tutkimuskysymyksistä johdetut konkreettiset tavoitteet voidaan esittää seuraavasti:

- Modulaarisen, uudelleenkäytettävän ja uudelleenkonfiguroitavan mallin määritteleminen tuotantolinjalle ja siihen liittyvälle tuotetiedolle.
- Tavoitteiden, vaatimusten ja menettelyjen määritteleminen ohjelmistojen versionhallinnalle.
- Ohjelmistokehitykseen ja versionhallintaan soveltuvien työkalujen ehdottaminen.
- Ohjelmistokehityksen edistämiseen liittyvien investointien ja toimenpiteiden ehdottaminen.

Tutkimuksen pääteemoja ovat ohjelmistojen modulaarisuus ja versionhallinta. Tutkimuksen painopisteenä on siis tutkittavan tuotantoprosessin informaatioprosessit ja niitä toteuttavien ohjelmistojen versionhallinta. Versionhallinta on rajattu vain ohjelmistoihin ja niiden kehittämiseen käytettäviin simulointimalleihin. Muun suunnitteluaineiston (kuten projektidokumentaatio ja fyysisten komponenttien tuotetieto) hallintaan on jo olemassa menettelyt ja työkalut. Simulaattorikehityksen ja versionhallinnan testaus eivät myöskään sisälly tutkimukseen. Rajaukset tehtiin, koska testausta ei ole mahdollista toteuttaa tutkimuksen aikataulun puitteissa.

## 1.4 Tutkimusmenetelmät

Tutkimuksessa käytettyjä tutkimusmenetelmiä ovat tapaukseen liittyvän dokumentaation ja kirjallisuuden analysointi sekä aktiivinen osallistuva havainnointi. Tapaukseen liittyvä

aineisto koostuu pääosin tilaajayrityksen tuottamista ja toimittamista teknisistä dokumenteista. Vapaamuotoiset keskustelut yrityksen edustajien kanssa ovat olleet myös tärkeä tiedonlähde tapauksen kontekstin ymmärtämisessä.

Tässä tutkimuksessa ei pyritä teoreettisten yleistyksien tekemiseen vaan tapauksen kokonaisvaltaiseen ymmärtämiseen. Tapaustutkimuksen lähestymistavaksi on valittu sekä poikittaissuuntainen että pitkittäissuuntainen läpileikkaus tutkittavasta tuotantolinjasta. Pitkittäissuuntaisen läpileikkauksen tarkoituksena on kuvata tuotantolinja kokonaisuudessaan korkealla abstraktiotasolla. Poikittaissuuntaisessa läpileikkauksessa, sen sijaan, valitaan hyvin rajattu ja edustava osakokonaisuus yksityiskohtaisemman tarkastelun kohteeksi. Poikittaissuuntaisella läpileikkauksella siis kuvataan tuotantolinjaa matalalla abstraktiotasolla. Tapaustutkimuksen tarkoituksena on tuottaa yksityiskohtaista tietoa, jonka avulla voidaan arvioida modernien ohjelmistokehityksen menetelmien soveltuvuutta teollisiin tuotekehysprojekteihin.

## 1.5 Tutkimuksen rakenne

Tutkimus jakautuu viiteen päälukuun. Ensimmäinen luku toimi johdatteluna tutkittavaan aiheeseen. Johdannossa perusteltiin siis tutkimuksen tarpeellisuutta, kuvattiin käsiteltävää tutkimusongelmaa ja siihen liittyviä rajauksia. Lisäksi esiteltiin tutkimuksen tavoitteet ja käytetyt tutkimusmenetelmät.

Toinen ja kolmas luku esittelevät aiheesta tehtyä aiempaa tutkimusta. Kirjallisuuskatsaus jakautuu kahteen osaan, joista ensimmäinen käsittelee konfiguraation- ja versionhallintaa sekä niiden yhteyttä elinkaariajatteluun. Kirjallisuuskatsauksen toinen osa keskittyy simulointiin, mallinnukseen ja tuotetiedon hallintaan. Kirjallisuuskatsauksen tarkoituksena on esitellä tämän tutkimuksen yhteyttä aiempiin tutkimuksiin ja perustella tutkimusongelmien muotoilun relevanttiutta.

Neljäs luku on tutkimuksen pääosio. Luvussa esitellään tapaustutkimuksen taustaa sekä tutkimuksen tuottamat tulokset analyyseineen. Tapaustutkimus jakautuu kolmeen osaan, joista ensimmäinen käsittelee tuotantolinjan modulaarista rakennetta. Seuraavassa osassa esitellään ohjelmistojen ja simulointimallien versionhallintaratkaisuja. Tapaustutkimuksen viimeisessä osassa laaditaan toimenpide-ehdotus ohjelmistokehityksen edistämiseksi tapaustutkimuksen havaintojen pohjalta. Toimenpide-ehdotus sisältää investointi- ja strategiaehdotuksia ohjelmistokehityksen edistämiseksi. Lisäksi käsitellään ohjelmistokehitystä tukevia työkaluja. Neljäs luku siis esittelee vastaukset asetettuihin tutkimusongelmiin.

Viimeinen luku on yhteenveto, jossa tutkimustulokset kootaan yhteen ja esitellään tulosten perusteella tehdyt päätelmät. Yhteenveto-osiossa myös arvioidaan tutkimukselle

asetettujen tavoitteiden saavuttamista ja ehdotetaan aiheeseen liittyviä jatkotutkimustarpeita.

# 2. KONFIGURAATION- JA VERSIONHALLINTA

## 2.1 Systeemin konfiguraationhallinta

Tässä luvussa tutustutaan konfiguraationhallintaan. Tarkoituksena on luoda yleiskuva konfiguraationhallintaprosessista. Ensimmäisessä alaluvussa käsitellään konfiguraationhallinnan käyttötarkoitusta ja toiminnallisuuksia. Toisessa ja kolmannessa alaluvussa tutustaan tarkemmin tämän tutkimuksen kannalta oleellisimpiin konfiguraationhallinnan osaprosesseihin, jotka ovat konfiguraation identifiointiprosessi ja muutostenhallintaprosessi.

### 2.1.1 Yleiskatsaus konfiguraationhallintaan

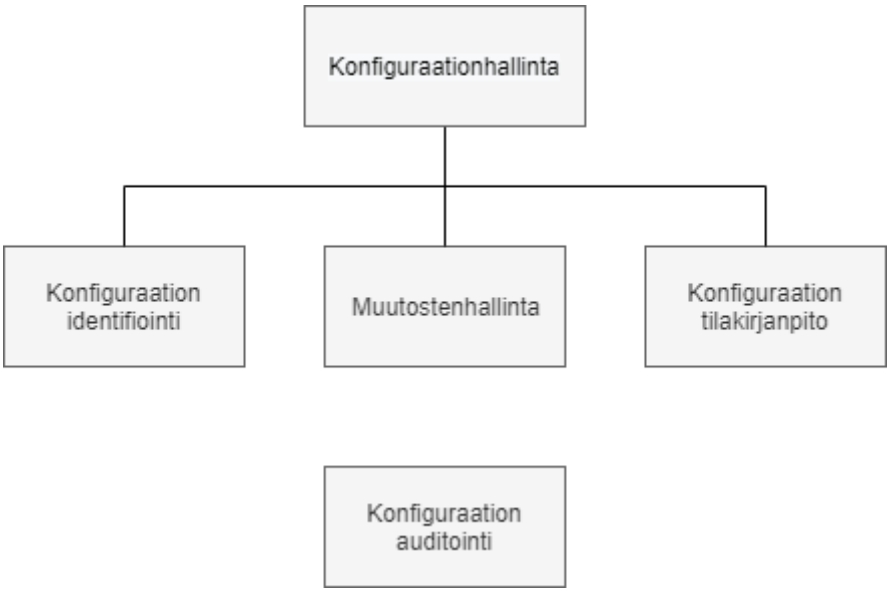
Tässä luvussa esitetyt konfiguraationhallinnan määritelmät perustuvat lähteeseen [1], ellei toisin mainita. Konfiguraationhallinta on systeemisuunnittelun lähestymistapa, jolla hallitaan kompleksisten systeemien muutosta. Hallittavat muutokset kohdistuvat sekä digitaaliseen tietoaineistoon (engl. data set) että siihen liittyvään reaalimaailman systeemiin [2]. Konfiguraationhallintatyökalun tarkoituksena on varmistaa systeemin yhtenäisyys ja vaatimustenmukaisuus, tietoaineiston ajantasaisuus sekä muutosten jäljitettävyys kaikissa systeemin elinkaaren vaiheissa.

Systeemin konfiguraatio muodostuu laitteistosta, ohjelmistosta ja niihin liittyvästä konfiguraatioinformaatiosta. Systeemin määrittelyt sekä fyysiset ja toiminnalliset ominaisuudet kuvaava tekninen dokumentaatio muodostavat konfiguraatioinformaation. Konfiguraatioinformaation avulla voidaan ylläpitää ajantasaista kuvausta laitokselle asennetusta laitteistosta ja ohjelmistosta sekä niiden muutoshistoriasta.

Tässä tutkimuksessa tiedolla viitataan englanninkieliseen käsitteeseen data. Tieto voidaan ajatella raaka-aineena, jota jalostamalla tuotetaan merkityksellisempää informaatiota. Tietoaineisto siis muuttuu informaatioksi, kun se esitetään jäsennellysti merkityksellisessä asiayhteydessä.

Konfiguraationhallintaa tukevia standardeja löytyy monia. Niistä osa on kuitenkin tarkoitettu aseteollisuuden käyttöön. Kansainvälisten kaupallisten standardien, ISO 10007 [3] ja IEEE 828 [4], mukaan konfiguraationhallinnan toimintoja ovat konfiguraation identifiointi, muutostenhallinta, tilakirjanpito ja auditointi.

Konfiguraationhallinnan toiminnot on esitetty kuvassa 1. Konfiguraation auditointi on esitetty muista toiminnoista erillään. Tämä havainnollistaa auditoinnin riippumattomuutta muista toiminnoista.




***Kuva 1.*** *Konfiguraationhallinnan perustoiminnot.*

Konfiguraation identifioinnin tehtävänä on määritellä systeemin rakenne sekä konfiguraationimikkeet ja niihin liittyvä konfiguraatioinformaatio. Muutostenhallinnalla varmistetaan, että systeemiin tehtävät muutokset toteutuvat vaatimustenmukaisesti. Muutostenhallintaprosessissa systeemiin kohdistuvat muutokset tunnistetaan ja dokumentoidaan. Lisäksi päätetään muutosehdotusten hyväksymisestä tai hylkäämisestä.
Tilakirjanpito voidaan suorittaa integroituna osana muutostenhallintaprosessia. Tilakirjanpidolla mahdollistetaan muutosten jäljitettävyys [3]. Siinä seurataan muutosehdotusten tilaa (kuten hyväksytty tai hylätty). Myös implementoitavien muutosten tiloista (kuten avoinna tai julkaistu) pidetään kirjaa.
Auditointi on riippumaton ja määrämuotoinen tarkastelu. Auditoinnilla varmistetaa*n*, että systeemin dokumentaatio vastaa nykytilannetta. Auditoinnin tehtävänä on myös varmistaa, että reaalimaailman systeemi täyttää sille asetetut fyysiset ja toiminnalliset vaatimukset.

### 2.1.2 Konfiguraation identifiointiprosessi

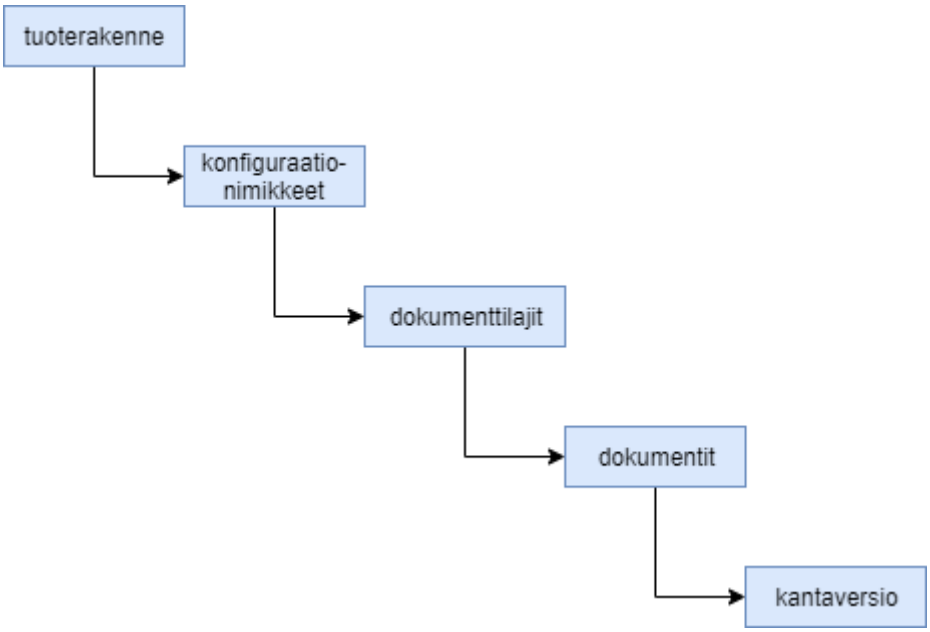
Konfiguraation identifiointiprosessi esitetään tässä luvussa lähteiden [3] ja [4] mukaan. Konfiguraation identifiointiprosessissa systeemille määritetään hierarkkinen rakenne, jossa systeemi jaetaan alisysteemeihin ja edelleen komponentteihin. Systeemi muodostuu ohjelmisto- ja laitteistokomponenteista sekä niiden välisistä yhteyksistä.

Toiminnallisen tai fyysisen erottelun avulla voidaan muodostaa modulaarinen systeemi. Modulaarisuus tukee uudelleenkonfigurointia ja uudelleenkäytettävyyttä. Tässä tutkimuksessa moduuliksi kutsutaan yhdestä tai useammasta komponentista muodostettua uudelleenkäytettävää yksikköä, jonka rajapinnat on määritelty täsmällisesti. Rajapinta on fyysinen tai toiminnallinen vuorovaikutus moduulien välillä.

Identifiointiprosessiin sisältyvät myös konfiguraatioon liittyvän informaation valinta ja yksilöinti. Konfiguraatioinformaatio sisältää tyypillisesti vaatimusmäärittelyn, rajapintojen määrittelyn, tekniset piirustukset, ohjelmiston lähdekoodin, osaluettelot, testimäärittelyt ja -ohjeet sekä käyttö- ja huolto-ohjeet. Informaatiolle määritellään yksilölliset tunnistetiedot sekä versiointikäytännöt jäljitettävyyden takaamiseksi.

Hyväksytty konfiguraatioinformaatio jäädytetään ja muodostetaan näin kantaversio (engl. baseline). Jäädytyksellä estetään hyväksytyn informaation muuttaminen. Kantaversio kuvaa siis tuotteen ominaisuuksia tiettynä ajankohtana ja toimii näin vertailukohtana tulevaisuuden kehitystyölle. IEEE:n mukaan kantaversio on formaalisti hyväksytty määrittely tai tuote, jota voidaan muuttaa vain formaalien muutoshallintamenettelyjen kautta.

Kantaversio koostuu yhdestä tai useammasta konfiguraationimikkeestä (engl. configuration item). Konfiguraationimike on konfiguraation hallittava yksikkö, jonka kompleksisuus vaihtelee. Konfiguraationimike voi olla esimerkiksi systeemi, moduuli, komponentti tai dokumentti. Konfiguraationimike on siis hierarkkinen rakenne, joka voi koostua useasta yksittäisestä nimikkeestä. Konfiguraationhallintaprosessissa konfiguraationimikettä käsitellään kuitenkin yhtenä itsenäisenä kokonaisuutena. Edellä kuvatun konfiguraation identifiointiprosessin vaiheet on esitetty kuvassa 2.




***Kuva 2.*** *Konfiguraation identifiointiprosessi.*

### 2.1.3 Muutostenhallintaprosessi

Muutostenhallinta on prosessi, jossa arvioidaan muutoksen mahdollisia vaikutuksia systeemiin. Muutosehdotuksen hyväksymis- ja hylkäämisperusteet määritetään riskianalyysin perusteella. Lisäksi huolehditaan hyväksyttyjen muutosten oikeanlaisesta toteuttamisesta ja systeemiä koskevan dokumentaation päivittämisestä. Tässä luvussa käsiteltävä muutostenhallintaprosessi esitetään ensisijaisesti Conradin (et al.) [5] tutkimuksen mukaan, ellei toisin mainita.

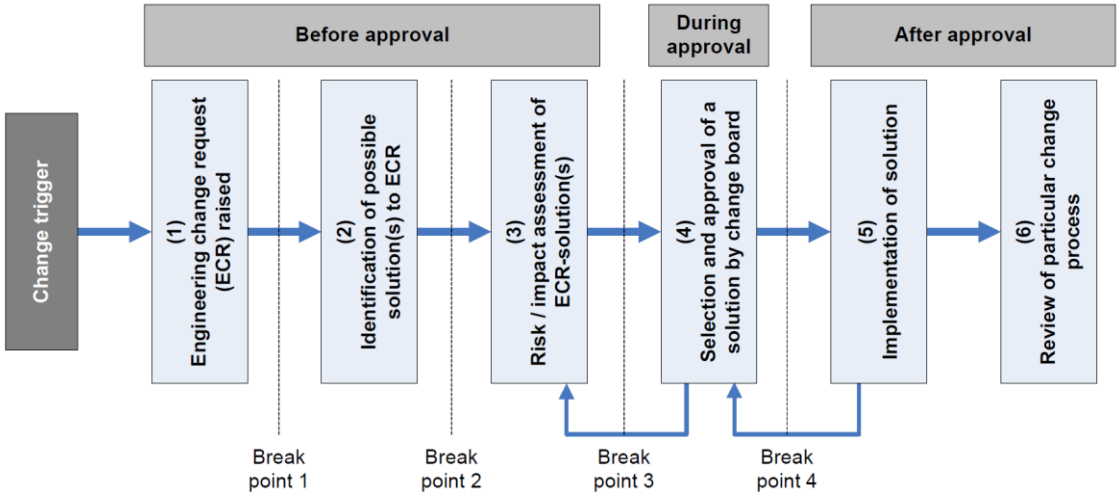
IEEE:n [4] määritelmän mukaan julkaisu on testattavien konfiguraationimikkeiden kokoelma, joiden käyttöönotto tapahtuu samanaikaisesti samassa fyysisessä ympäristössä. Julkaistuun kantaversioon ja sen konfiguraationimikkeisiin kohdistuvat muutokset vaativat hallintaa [3]. Muutostenhallintaa sovelletaan vain konfiguraationimikkeille, joiden kantaversio on jo olemassa [4].

Conrad (et al.) esittelee tutkimuksessaan geneerisen muutostenhallintaprosessin. Prosessi koostuu kuudesta vaiheesta, joista ensimmäinen on hyvin perustellun muutospyynnön esittäminen. Muutospyynnön voi esittää esimerkiksi suunnittelija tai asiakas. Seuraavassa vaiheessa pyritään tunnistamaan potentiaalisia ratkaisuja muutospyynnön toteuttamiseksi.

Kolmas vaihe on muutoksen vaikutusten ja siihen liittyvän riskin arvioiminen. Eräs riskien ja vaikutusten arvioinnissa yleisesti käytetty menetelmä on vika- ja vaikutusanalyysi (engl. Failure Mode and Effect Analysis). Analyysi perustuu seuraaviin kysymyksiin:

- “Mitä tapahtuu, jos pyydetty muutos toteutetaan?”
- ”Miten negatiiviset vaikutukset saadaan selville ja miten ne ehkäistään?”

Edellä mainittuihin kysymyksiin saatujen vastausten perusteella valitaan ratkaisu, jolla on alhaisin riski ja vähiten sivuvaikutuksia muihin osasysteemeihin. Ratkaisun valitsee ja hyväksyy muutoslautakunta. Muutoslautakunta voi koostua esimerkiksi tuotepäälliköistä, pääinsinööreistä ja tuotekehityspäälliköstä.
Hyväksytty ratkaisu implementoidaan sovitun aikataulun mukaisesti. Sekä hyväksytyt että hylätyt muutokset dokumentoidaan. Lopuksi voidaan näin arvioida muutosprosessin tuloksellisuutta ja oppia mahdollisista virheistä. Muutostenhallintaprosessi on esitetty kuvassa 3.




***Kuva 3.*** *Muutostenhallintaprosessi [5].*

## 2.2 Ohjelmistojen versionhallinta

Tässä luvussa käsitellään ohjelmistojen versionhallintaa. Muutostenhallinta kuvaa menettelyä, jolla hallitaan muutosehdotuksia ja muutosten vaikutuksia sekä reaalimaailman systeemin että siihen liittyvän digitaalisen tietoaineiston näkökulmasta. Versionhallinta taas on menettely, jolla seurataan tiedostoihin tai kansioihin tehtyjä muutoksia [6]. Toisin kuin muutostenhallinnassa, versionhallinnassa muutokset kohdistuvat siis vain digitaaliseen tietoaineistoon. Ensin määritellään versionhallinnan käsite ja tehtävät sekä ohjelmistojen versionhallinnassa yleisimmin käytetyt kantaversiot. Tämän jälkeen tutustutaan erilaisiin versiointimalleihin ja automaatio-ohjelmistojen versionhallintaan liittyviin erityispiirteisiin. Lopuksi käsitellään ohjelmistojen modularisointia ja sen liiketoimintavaikutuksia.

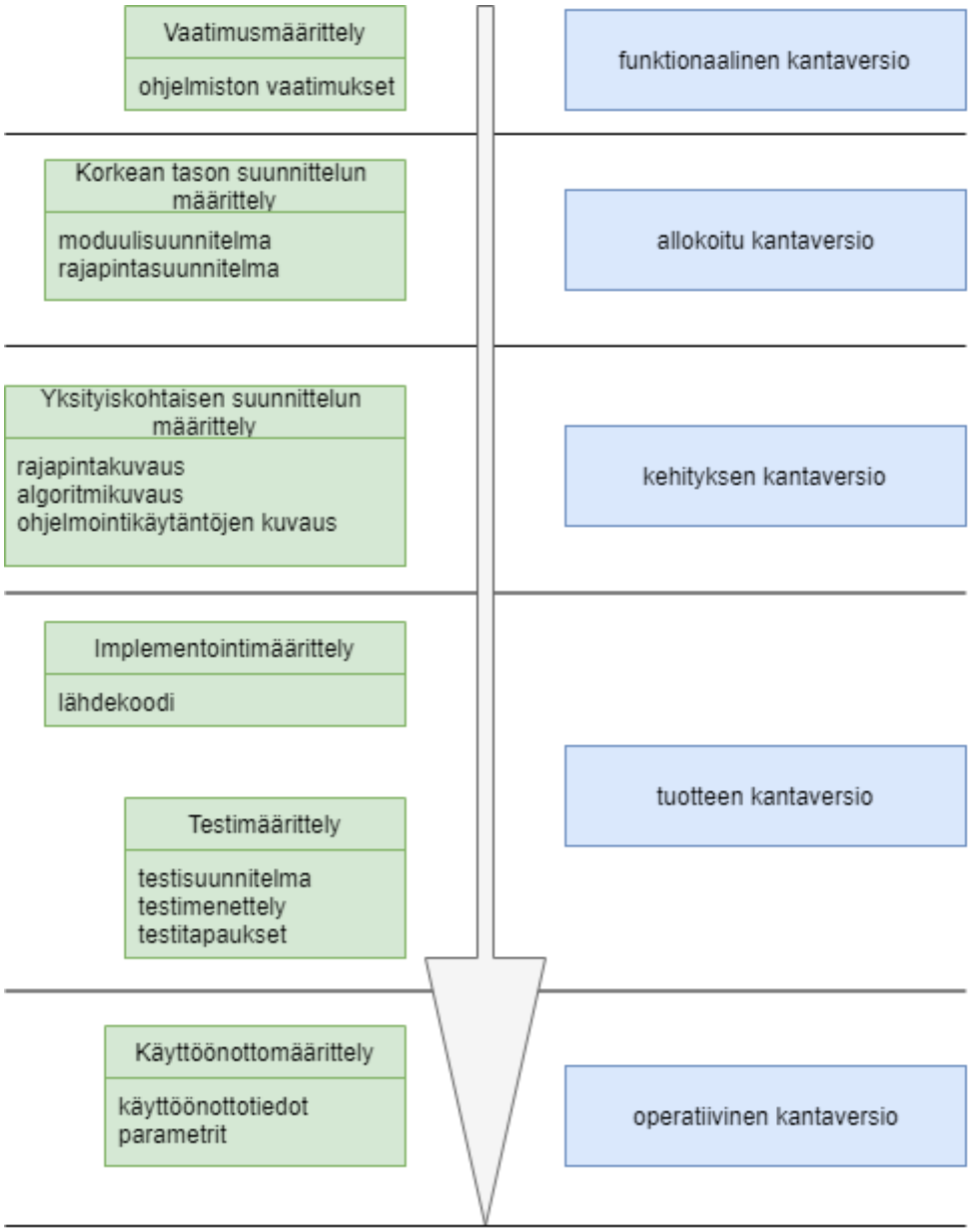
### 2.2.1 Kantaversiot

Versionhallinta voidaan ajatella ohjelmistojen konfiguraationhallinnan osa-alueeksi. Tässä tutkimuksessa käsitteellä versionhallinta tarkoitetaan konfiguraationimikkeisiin ja

niistä koostuviin kantaversioihin kohdistuvien muutoksien seurantaa. Ohjelmiston konfiguraationimikkeitä ovat esimerkiksi ohjelmistomoduulit ja -komponentit sekä niihin liittyvät dokumentit.

IEEE:n [1] määritelmän mukaan versionhallinnan tehtäviä ovat kantaversioiden perustaminen ja ylläpitäminen sekä kantaversioihin tehtyjen muutosten tunnistaminen ja hallitseminen. Muutoksia on hallittava siten, että tietyn hetken kantaversiosta on mahdollista palata takaisin edelliseen kantaversioon. Kantaversio määrittelee siis jaksottaisen alkupisteen uusille muutoksille [6].

Bendixin & Borraccin [7] mukaan kantaversiot parantavat uudelleenkäytettävyyttä ja vähentävät ohjelmointivirheiden määrää. Pressmanin [8] mukaan kantaversio merkitsee ohjelmistokehityksen virstanpylvästä. Kuten luvussa 2.1.2 todettiin, uusi kantaversio voidaan hyväksyä vain muutostenhallintamenettelyn kautta. Kuvassa 4 on esitetty tavanomaisesti käytettyjä kantaversioita ja niihin liittyviä konfiguraationimikkeitä ohjelmistokehityksen elinkaaren eri vaiheissa. Kuvassa ohjelmistokehityksen etenemissuunta on osoitettu nuolella. Kuvantiedot perustuvat lähteisiin [8] ja [9].



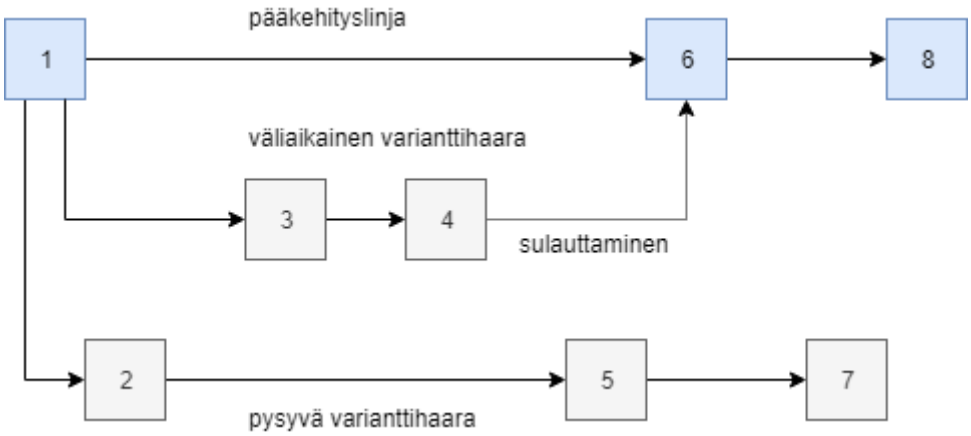

***Kuva 4.*** *Tavanomaisia kantaversioita, mukailtu lähteistä [8] ja [9].*

### 2.2.2 Versiointi

Tässä luvussa esitelty versiointi perustuu ensisijaisesti lähteeseen [6], ellei erikseen toisin mainita. Versiointi on ohjelmistoprojektin tuotosten ja niiden historiatietojen hallintaa. Versiointi seuraa konfiguraationimikkeiden historiallisia yhteyksiä. Nämä yhteydet muodostavat rakenteen, jota kutsutaan versiograafiksi. Versio, revisio ja variantti ovat oleellisia versiograafiin liittyviä käsitteitä. IEEE:n [1] mukaan versio on ohjelmiston konfiguraationimikkeen julkaisu. Revisiot ovat historiallisesti peräkkäisiä versioita ja variantit rinnakkaisia versioita. Variantit muodostavat haaroja (engl. branch) kehityslinjaan [10]. Variantit eroavat toisistaan toimintojen, suunnittelun tai implementoinnin näkökulmasta.

Useiden varianttien sisältämien muutosten yhdistämistä uudeksi versioksi kutsutaan sulauttamiseksi (engl. merge).

Kuvassa 5 on esimerkki ohjelmistokehityksen versiograafista. Versiot on järjestetty ajan mukaan juoksevalla versionumerolla. Pääkehityslinjan versiot on merkitty sinisellä värillä ja variantit harmaalla.




***Kuva 5.*** *Esimerkki versiograafista.*

Edellä kuvattua revisioihin, variantteihin ja sulauttamiseen perustuvaa versiointimallia kutsutaan klassiseksi versioinniksi. Versiointimallia, jossa konfiguraationimikkeen versio muodostetaan lisäämällä haluttu muutosjoukko (engl. change set) kantaversioon, kutsutaan muutosjoukkoversioinniksi. Muutosjoukkoversioinnissa muutokset tallennetaan itsenäisinä deltoina, joilla tarkoitetaan tiedostoissa tai versioissa tapahtuneita varsinaisia muutoksia.

Muutosjoukkoversioinnin lähestymistapana on ohjelmiston ominaisuuksien (engl. feature) kuvaus (engl. mapping) muutosjoukkoina. Tällöin ohjelmisto voidaan konfiguroida valitsemalla sopiva ominaisuuksien osajoukko. Muutosjoukot järjestellään ja lisätään uusiin ohjelmistoversioihin luomisaikojensa mukaisessa järjestyksessä. Päällekkäisten deltojen tapauksessa, uudempi muutos syrjäyttää vanhemman [10]. Muutosjoukkoihin perustuva versiointi ei kuitenkaan toimi hyvin käytännössä, sillä päällekkäiset deltat aiheuttavat konflikteja. Tällöin tuloksena saattaa olla toimimaton tai virheellisesti toimiva koontiversio (engl. build). Lisäksi laajoissa ohjelmisto projekteissa yksittäisten ohjelmistoversioiden luotettava tunnistaminen muuttuu vaikeaksi muutosten suuren määrän vuoksi.

Käytännöllisempi lähestymistapa on yhdistellä klassisen versioinnin ja muutosjoukkoversioinnin lähestymistapoja. Tällöin ohjelmiston mahdollisten permutaatioiden lukumäärää voidaan vähentää perustamalla useita uusia kantaversioita kehitysprojektin aikana. Tällöin ohjelmistoversio koostuu kantaversionsa lisäksi vain muutamasta lisämuutoksesta.

Kun deltoja yhdistellään vain kantaversioidensa kanssa, muutosjoukkoversiointi redusoituu klassiseksi versioinniksi. Tämä yksinkertaistaa versioiden valitsemista ja edistää näin versioinnin luotettavuutta. Kantaversiot helpottavat myös varianttien hallintaa ja ylläpidettävyyttä [7].

### 2.2.3 Automaatio-ohjelmistojen versionhallinnan erityispiirteet

Tässä luvussa esitellään teollisuusautomaatiossa käytettyjen ohjelmistojen erityispiirteitä, jotka luovat omat haasteensa ja vaatimuksensa versionhallinnalle. Erityisesti keskitytään ohjelmoitaville logiikoille (engl. Programmable Logic Controller, PLC) kehitettävien ohjelmistojen versionhallintaan. Lisäksi käsitellään tällaisten ohjelmistojen kehitykseen ja testaukseen käytettävien simulointimallien versionhallintaa.

Binääritiedostot ovat tiedostoja, jotka eivät ole selväkielisiä tekstitiedostoja. Tällaiset tiedostot voivat sisältää esimerkiksi graafisia elementtejä, ääniä tai pakattuja kansioita. Lohkokaavioita hyödyntävien PLC-ohjelmointiympäristöjen (kuten Siemensin Step 7) ja testaukseen soveltuvien simulointiympäristöjen (kuten MathWorksin Simulink) tuotokset ovat tyypillisesti binääritiedostoja. Binääritiedostojen versionhallintaan on siis syytä kiinnittää huomiota.

Hajautettu versionhallintatyökalu perustuu vertaisverkkoon, jossa käyttäjät jakavat projekteja tai niiden muutoksia keskenään. Tällöin kukin paikallinen kopio projektista sisältää kokonaisvaltaisen versiohistorian. O'Sullivanin [11] mukaan binääritiedostojen versionhallinta eroaa tekstitiedostoista siinä, että binääritiedostojen versioita ei voi sulauttaa eikä binääritiedostojen versioiden välisiä eroja ole helppo tunnistaa. Näin ollen hajautetut versionhallintatyökalut (kuten Git) eivät sovellu kovin hyvin binääritiedostojen hallintaan. Toistuvasti muokattaessa, binääritiedostojen vaatima tallennustilan määrä saattaa kasvaa nopeasti suhteessa sulautettavissa oleviin tekstitiedostoihin. Estublier'n (et al.) [6] mukaan kuitenkin monet modernit ohjelmistojen versionhallintatyökalut käyttävät ZIP-tyyppistä pakkaustekniikkaa. Tämä johtuu siitä, että deltojen käyttäminen on menettänyt merkitystään levytilan edullisuuden ja binääritiedostojen yleisyyden vuoksi. Toisin kuin hajautetulla työkalulla, keskitettyyn palvelimeen perustuvalla työkalulla voi ladata vain tietyn version sisältävän paikallisen kopion [11]. Suurten binääritiedostojen hallintaan soveltuu siis paremmin keskitettyä palvelinta käyttävä versionhallintatyökalu.

PLC-ohjelmistojen vahva alustariippuvuus hankaloittaa niiden versionhallintaa. Vogel-Heuser (et al.) [12] toteaa tutkimuksessaan, että eri valmistajien alustojen väliset erot tekevät ohjelmistojen vastaavuuksien varmistamisesta haastavaa. Versioiden ylläpitoa vaikeuttaa myös laite- (engl. machine) ja laitosvarianttien suuri määrä. Ylläpitoa voidaan kuitenkin helpottaa modularisoimalla ohjelmistoja.

Jenkinsin (et al.) [13] mukaan automaatio-ohjelmistoilla ohjataan ja suojataan reaaliaikajärjestelmiä, joilta vaaditaan jatkuvaa käytettävyyttä. Täten toleranssi systeemin käyttökatkoille (engl. system downtime) on hyvin alhainen. Versionhallinnan haasteena on tällöin ohjelmistopäivitysten yhteydessä ilmenevät ongelmat. Käytettävyyden takaamiseksi, ohjelmisto on kyettävä palauttamaan takaisin viimeisimpään toimivaan versioon. Tällöin systeemi pysyy käytettävissä myös ohjelmointivirheiden ja muiden ongelmien tunnistamisen aikana. Viimeisimmän toimivan version palauttamiseen voidaan käyttää esimerkiksi kuvan 4 mukaista operatiivista kantaversiota.

Automaatio-ohjelmistoille on tyypillistä, että niissä (esimerkiksi parametrien asetteluarvoja on muuteltava käyttöönotossa. Nämä viime hetken muutokset tapahtuvat yleensä laitoksella (engl. on-site) [12]. Käyttöönotetut asetteluarvot tulee siis päivittää uuteen ohjelmistoversioon [13]. Haasteeksi muodostuu tällöin käyttöönotossa tehtyjen muutosten vieminen versionhallintaan. Jos muutoksia ei dokumentoida, versionhallinnassa ja laitoksella voi olla eri ohjelmistoversiot.

Simulointimalleja voidaan käyttää esimerkiksi säätöalgoritmien suunnitteluun sekä säädinten viritykseen ja testaukseen. Simulointiprojekti voi sisältää useita simulaatioita, joissa testitapauksia tai itse simulointimallia varioidaan [14]. Ohjaussovelluksien toteutuksissa ja hallittavien systeemien malleissa voi olla myös asiakaskohtaisia eroavaisuuksia. Varianttien hallinta on siis ratkaisevassa osassa versionhallinnan toimivuuden kannalta. 3D-mallien (engl. three dimensional, 3D) simulointi tuottaa suuren määrän erityyppisiä tiedostoja. Samaan konfiguraatioon liittyvien tiedostojen välisiin riippuvuuksiin tulee siis kiinnittää erityistä huomiota. Walkerin (et al.) [15] mukaan hyväksi havaittu käytäntö on erottaa simulointimallit ja niiden parametrit toisistaan. Tällöin simulointiprojektin konfiguraatioon sisällytetään parametrien konfigurointitiedostot.

Novákin (et al.) [16] mukaan simulointimallien parametrit voidaan jakaa simulointiparametreihin ja simuloinnin ajoparametreihin (engl. simulation run parameter). Simulointiparametrit kuvaavat laitteiden fyysisiä ominaisuuksia, kuten pituus tai massa. Simuloinnin ajoparametrit taas ovat tiettyyn simulointikokeeseen liittyviä parametreja, kuten simulointiaika tai alkutilat (engl. initial conditions). Simulointiparametrit ovat siis matemaattisia vakioita, jotka määrittelevät mallin dynamiikan.

Seuraavassa alaluvussa käsitellään ohjelmistojen modularisointia. Modulaariset tuoteperheet helpottavat varianttien hallintaa ja nopeuttavat ohjelmistojen kehitystyötä. Varianttien hallinta parantaa ohjelmistojen ylläpidettävyyttä ja versioinnin luotettavuutta.

### 2.2.4 Ohjelmistotuoteperheet

Tässä luvussa ohjelmistotuoteperheet esitellään pääosin Lindenin (et al.) [17] ja Breivoldin (et al.) [18] tutkimusten mukaan, ellei jäljempänä toisin ilmaista. Ohjelmistotuoteperhe on samankaltaisista ohjelmistoista koostuva kokonaisuus, portfolio. Samankaltaisten ohjelmistotuotteiden muodostamasta perheestä käytetään kirjallisuudessa myös nimitystä ohjelmistotuotelinja. Modulaariset ohjelmistotuoteperheet tukevat uudelleenkäytettävyyttä ohjelmistokehityksessä. Lindenin (et al.) mukaan tämä lähestymistapa alentaa kehitys- ja ylläpitokustannuksia sekä lyhentää tuotekehitykseen kuluvaa aikaa (engl. time-to-market).

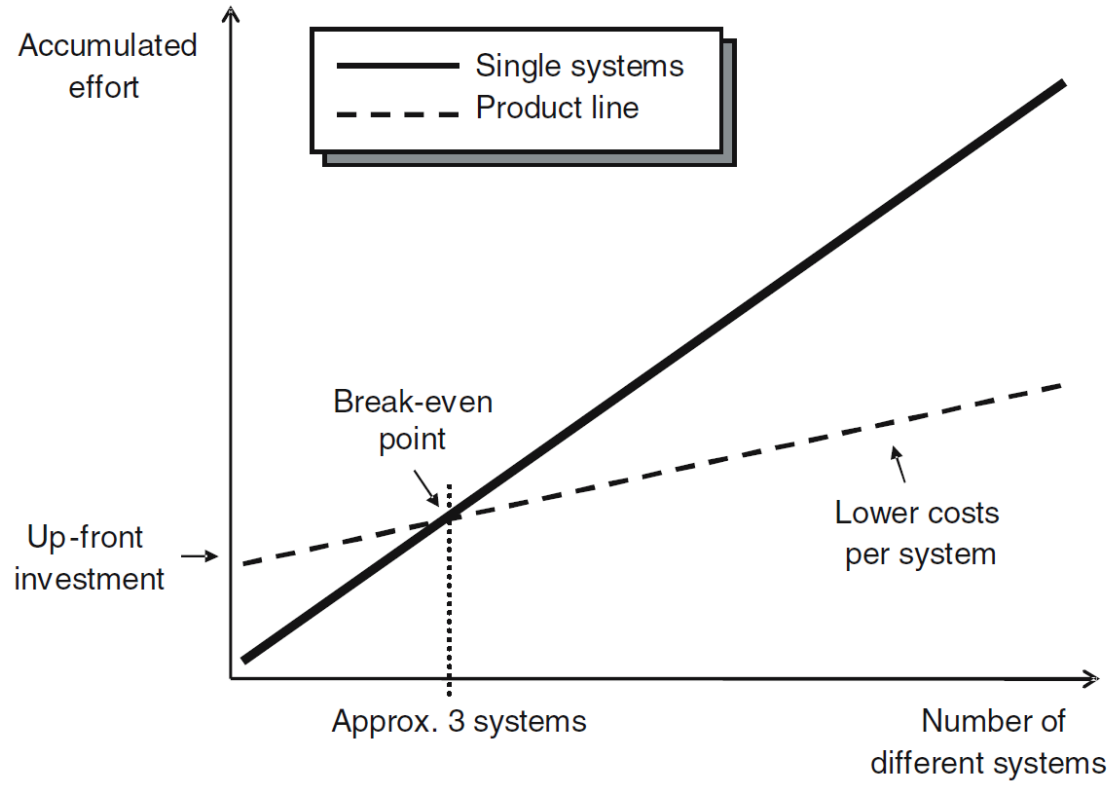
Fischerin (et al.) [19] mukaan prosessiteollisuudessa PLC-ohjelmistojen systemaattinen uudelleenkäyttö on harvinaista. Ohjelmistomoduuleita tai -komponentteja ei siis juurikaan käytetä. Prosessiteollisuudessa ohjelmistojen uudelleenkäyttö perustuu tavanomaisesti ”kopioi, liitä ja muokkaa” –menetelmään, jonka käyttäminen johtaa suureen määrään variantteja. Menetelmä myös heikentää ohjelmistojen ymmärrettävyyttä ja ylläpidettävyyttä. Versioiden hallitsemiseminen on tällöin vaikeaa ja monimutkaista. Lindenin (et al.) mukaan perinteiseen uudelleenkäyttöön (kuten vanhojen ohjelmistojen muokkaus) verrattuna tuoteperheiden käyttö voi alentaa ohjelmiston kokonaiskustannuksia jopa 90 %.

Systemaattinen uudelleenkäyttö perustuu tuotevarianttien hallintaan. Variantteja hallitaan määrittelemällä ja mallintamalla tuoteperheen tuotteiden väliset yhteneväisyydet ja eroavaisuudet. Näin saadaan niin kutsuttu ominaisuusmalli (engl. feature model). Fischerin (et al.) [19] mukaan modulaarinen ohjelmisto voidaan kehittää ominaisuusperustaisen ohjelmoinnin (engl. feature-oriented programming) menetelmällä. Tällöin ohjelmisto koostuu kaikille tuotevarianteille yhteisistä ohjelmistomoduuleista ja tietylle tuotevariantille parametrisoiduista ohjelmistomoduuleista. Näin ollen parametrit määrittävät tiettyä tuotevarianttia koskevat toiminnalliset ominaisuudet.

Linden (et al.) jaottelee ohjelmistotuotteiden ominaisuudet kolmeen päätyyppiin, jotka ovat yhteneväinen, vaihteleva ja tuotekohtainen ominaisuus. Yhteneväiset ominaisuudet ovat kaikille tuotteille yhteisiä toiminnallisia tai ei-toiminnallisia ominaisuuksia. Ei-toiminnallinen ominaisuus voi liittyä esimerkiksi tuotteen luotettavuuteen tai suorituskykyyn. Vaihtelevat ominaisuudet ovat vain valituille tuotteille yhteisiä ominaisuuksia ja tuotekohtaiset ominaisuudet nimensä mukaisesti vain yhteen tuotteeseen liittyviä ominaisuuksia. Tuotekohtaiset ominaisuudet voivat määräytyä esimerkiksi asiakasvaatimusten perusteella.

Lindenin (et al.) ja Breivoldin (et al.) mukaan ohjelmistotuoteperheisiin siirtymiseen vaadittava investointi on noin kolminkertainen suhteessa yksittäiselle systeemille kehitettävän ohjelmiston kustannuksiin. Toisin sanoen, investoinnin break-even –piste saavute-

taan kolmen ohjelmiston kehitystä vastaavan työmäärän jälkeen. Alkuinvestointi muodostuu muun muassa ohjelmistomoduulien kehittämiseen ja organisaatiomuutoksiin liittyvistä kustannuksista. Kuvassa 6 on havainnollistettu investoinnin taloudellista kannattavuutta.




***Kuva 6.*** *Tuoteperheiden käytön taloudellisuus [17].*

Lindenin (et al.) mukaan tapaustutkimukset ovat osoittaneet, muun muassa, seuraavia ohjelmistotuoteperheiden liiketoimintahyötyjä:

- kehitysajan lyheneminen alle puoleen (50 %) alkuperäisestä
- kalibrointi ja ylläpitokustannusten aleneminen viidenneksellä (20 %)
- laatukustannusten aleneminen.

Breivoldin (et al.) mukaan siirtymä kohti modulaarisia ohjelmistotuoteperheitä kannattaa tehdä pienin askelin. Tällöin voidaan minimoida vaadittava alkuinvestointi ilman haittavaikutuksia muihin meneillään oleviin projekteihin. Tuotekehitystiimin tulee myös tehdä tiiviisti yhteistyötä implementointitiimin kanssa siirtymäkauden aikana, jotta saavutetaan yhteisymmärrys siirtymän tavoitteista.

## 2.3 Elinkaari

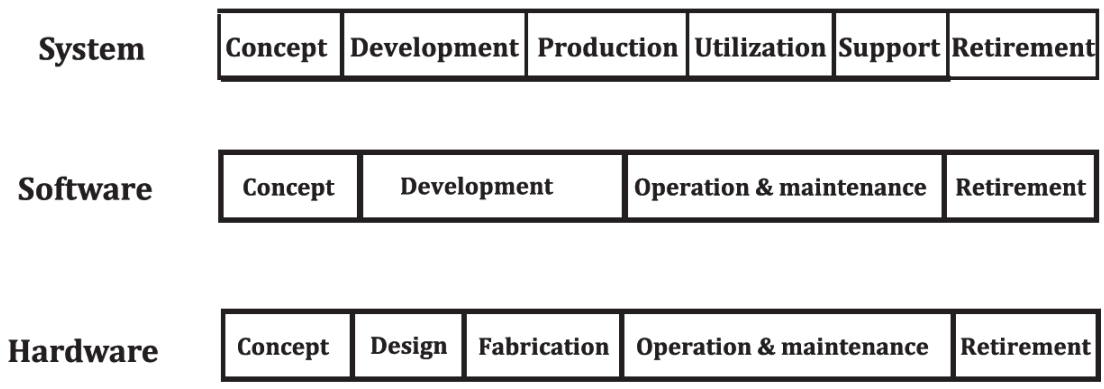
Konfiguraation- ja versionhallinta ovat prosesseja, jotka tukevat systeemin elinkaaren hallintaa. Tässä luvussa tutustutaan elinkaariajatteluun. Aluksi esitellään systeemiin, laitteistoon ja ohjelmistoon liittyviä elinkaarimalleja. Tämän jälkeen tutustutaan tuotteen elinkaareen ja sen hallintaan. Lopuksi käsitellään automaatio-ohjelmiston kehitykseen liittyvää elinkaarta.

### 2.3.1 Elinkaarimallit

Luvuissa 2.1 ja 2.2 käsiteltyjä konfiguraationhallintaa ja versionhallintaa sovelletaan läpi systeemin koko elinkaaren. Tässä luvussa käsitellään systeemin, laitteiston ja ohjelmiston elinkaarimallien välisiä eroja sekä näistä eroista aiheutuvia haasteita systeemin elinkaaren hallinnan kannalta. Systeemin ja ohjelmiston elinkaaren hallintaa käsittelevän standardin, IEEE 24748 [20], määritelmän mukaan elinkaari kuvaa ihmisen luoman kokonaisuuden (kuten systeemi, projekti, tuote tai palvelu) kehityskulkua alkaen konseptoinnista ja päättyen käytöstä poistamiseen. Jenkinsin (et al.) [13] mukaan elinkaari on ajanhetki, jolloin uusi komponenttiversio tulee saataville. Elinkaarella ei siis ole ajanhetki, jolloin tietty systeemin komponentti on vaihdettava. Komponentti tulee vaihtaa viimeistään sen teknisen käyttöiän päättyessä.

Systeemin elinkaaren vaiheita ovat konseptointi, kehitys, tuotanto, käyttö, ylläpito ja käytöstä poistaminen. Systeemin elinkaarimalli ei sellaisenaan sovellu alakohtaiselle (engl. domain-specific) alisysteemille (kuten ohjelmisto tai laitteisto). Kutakin alakohtaista alisysteemiä tulee siis käsitellä yksittäisenä kokonaisuutena, jolla on oma elinkaarensa.

Kuvassa 7 on havainnollistettu systeemin, ohjelmiston ja laitteiston elinkaarimallien eroavaisuuksia. Huomion arvoista on, että elinkaarimallien aikaskaalat eivät ole samoja. Mallien eliniät tai elinkaarien vaiheiden pituudet eivät siis ole vertailukelpoisia keskenään [20].




***Kuva 7.*** *Systeemin, ohjelmiston ja laitteiston elinkaarimallit [20].*

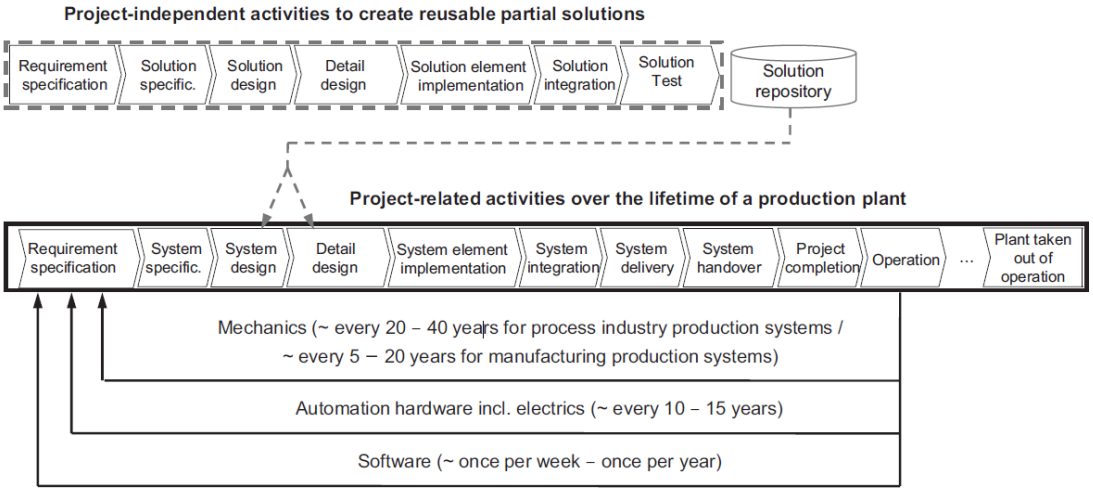
Vogel-Heuserin (et al.) [21] mukaan automatisoidun tuotantosysteemin elinkaaren hallinnan haasteena on erilaisten alakohtaisten elinkaarien yhtäaikainen hallitseminen. Automatisoitu tuotantosysteemi on eräänlainen mekatroninen systeemi. Tällainen systeemi koostuu mekaanisista osista, sähkö- ja elektroniikkaosista sekä ohjelmistosta.

Automatisoidun tuotantosysteemin elinikä on tyypillisesti useiden vuosikymmenien pituinen. Systeemin komponentteihin (kuten anturit, toimilaitteet ja ohjelmistokomponentit) kohdistuvat vaatimukset ja toiminnallisuudet muuttuvat todennäköisesti systeemin eliniän aikana. Systeemin kehitysvaiheessa on siis otettava huomioon mahdollisuus mukauttaa toiminnallisuuksia eliniän aikana. Systeemin kehityskulkua tuleekin suunnitella, toteuttaa ja dokumentoida systemaattisella tavalla ja siten varmistaa systeemin ylläpidettävyys sen koko eliniän ajan.

Fyysiset komponentit kuluvat ja ikääntyvät käytön aikana. Komponentteja on siis vaihdettava ajoittain. Toimenpidettä kutsutaan modernisoinniksi. Sähkö- ja elektroniikkakomponenteilla on lyhyempi elinkaari kuin mekaanisilla komponenteilla.

Jenkinsin (et al.) [13] mukaan teknologian kehittyessä myös ohjelmistoihin vaaditaan yhä useammin päivityksiä. Syitä ohjelmistojen päivittämiseen ovat esimerkiksi uusien ominaisuuksien tarjoaminen, suorituskyvyn ja luotettavuuden parannukset sekä kyberturvallisuuteen liittyvien haavoittuvuuksien paikkaaminen. Ohjelmiston elinkaari on siis lyhyempi kuin fyysisten komponenttien.

Erilaisten elinkaarien vuoksi, automatisoidun tuotantosysteemin muutostenhallintaan ja versiointiin on kiinnitettävä erityistä huomiota. Ohjelmistojen näkökulmasta, elinkaaren hallinnan haasteita ovat ohjelmistojen ylläpidettävyyden ja laajennettavuuden takaaminen [21]. Kuvassa 8 on esitetty tuotantolaitosprojektin elinkaaren vaiheet sekä alakohtaisten (mekaniikka, sähköautomaatio ja ohjelmisto) elinkaarien pituudet.

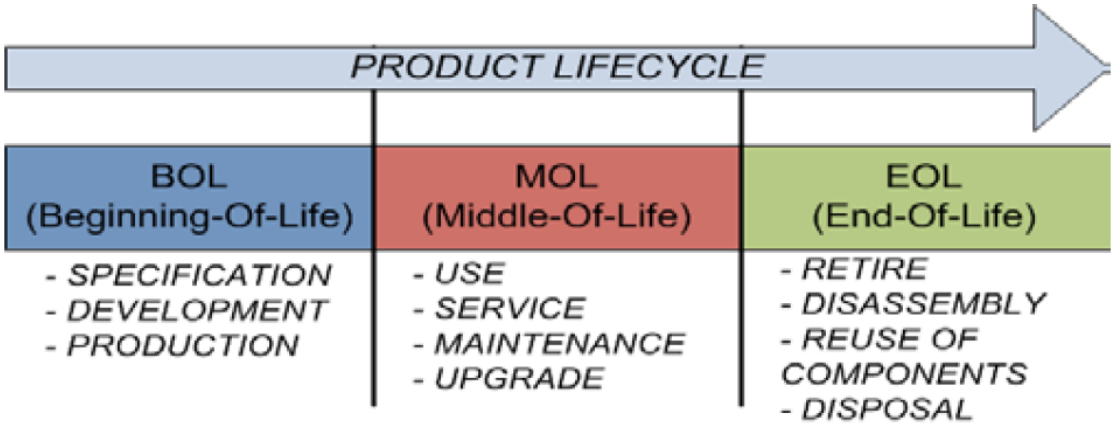



***Kuva 8.*** *Tuotantolaitoksen elinkaari ja modernisointi [21].*

### 2.3.2 Tuotteen elinkaari

Tuotteena voidaan pitää mitä tahansa markkinoille tarjottavaa hyödykettä, jolla pyritään tyydyttämää jokin asiakkaan tarve. Tuote voi siis olla aineellinen (kuten fyysinen esine) tai aineeton (kuten ohjelmisto tai algoritmi). Tuotteen elinkaaren hallinta (engl. Product Lifecycle Management, PLM) on systemaattinen tapa, jolla pyritään hallitsemaan tuoteinformaatiota tuotteen koko elinkaaren ajan. Tuoteinformaatio voi liittyä itse tuotteeseen, sen työvaiheisiin tai toimitusprosessiin. Tuotteen elinkaaren hallinta ei siis ole ohjelmisto tai metodi vaan konsepti tai ajattelutapa. Tuotteen elinkaaren vaiheita ovat tutkimus, kehitys, suunnittelu, tuotanto, käyttö ja ylläpito sekä kierrätys tai tuhoaminen [22, 23].

Lin (et al.) [23] mukaan tuotteen elinkaari voidaan jakaa kolmeen ajanjaksoon, jotka ovat elinkaaren alkuosa, keskiosa ja loppuosa. Elinkaaren alkuosa (engl. beginning of life) muodostuu tuotteen konseptoinnista suunnittelusta ja toteutuksesta. Keskiosa (engl. middle of life) pitää sisällään tuotteen käytön, ylläpidon ja päivittämisen. Elinkaaren loppuosa (engl. end of life) on ajanjakso, jolloin tuote poistetaan käytöstä kierrättämällä tai hävittämällä. Kuvassa 9 on havainnollistettu edellä kuvattua tuotteen elinkaaren jaottelua.




***Kuva 9.*** *Tuotteen elinkaari [24].*

Deuterin (et al.) [25] mukaan käsitteen PLM yhteydessä tuotteella viitataan kirjallisuudessa fyysisiin tuotteisiin, kuten laitteistoon. Ohjelmistoihin sen sijaan viitataan käsitteellä sovelluksen elinkaaren hallinta (engl. Application Lifecycle Management, ALM). Deuter toteaa myös, ettei PLM sellaisenaan sovellu ohjelmistojen kehitykseen. Sama pätee myös toisin päin eli ALM ei sovellu tuotteeseen liittyvän laitteiston elinkaaren hallintaan. Bricognen (et al.) [26] mukaan käsitteellä ALM tarkoitetaan ohjelmistokehityksen toimintojen koordinointia ja kehitysprosessin tuotosten (kuten vaatimukset, lähdekoodi ja testitapaukset) hallintaa läpi ohjelmistotuotteen elinkaaren. Seuraavassa alaluvussa käsitelläänkin automaatiosovelluksen kehityksen elinkaarta.

### 2.3.3 Automaatiosovelluksen kehityksen elinkaari

Tässä luvussa esitellään automaatiosovelluksen kehityksen elinkaari määrittelystä käyttöönottoon eli ohjelmistotuotteen elinkaaren alkuosa. Tässä tutkimuksessa noudatetaan Thramboulidis'n [27] määritelmää, jonka mukaan automaatiosovellus on ohjelmisto, joka sisältää tietyn systeemin hallintaan käytetyn ohjauslogiikan. Tässä luvussa esitellään automaatiosovelluksen kehityksen elinkaari ensisijaisesti Dubeyn [28] tutkimuksen mukaan, ellei erikseen toisin ilmaista.

Automaatiosovelluksen kehitysprojekteissa tuotetaan sovelluksia, joilla valvotaan ja ohjataan kompleksisia systeemejä. Automaatiosovelluksen kehityksen elinkaari voidaan jakaa neljään päävaiheeseen: vaatimusten määrittely ja suunnittelu, sovelluksen kehitys ja testaus sekä käyttöönotto. Käyttöönottoa seuraa ylläpitovaihe, jota ei käsitellä tässä alaluvussa.

Vaatimusten määrittelyn ja suunnittelun tarkoituksena on systeemin toimintaperiaatteiden kuvaileminen. Näin voidaan tuottaa informaatiota, joka helpottaa käytettävän laitteiston (kuten kenttälaitteet ja säätimet) valitsemista. Kehitysvaiheessa tapahtuu varsinainen sovelluskehitys, jossa tuotetaan ohjauslogiikka säätimelle (esim. PLC). Kehitysvaihetta seuraa testausvaihe.

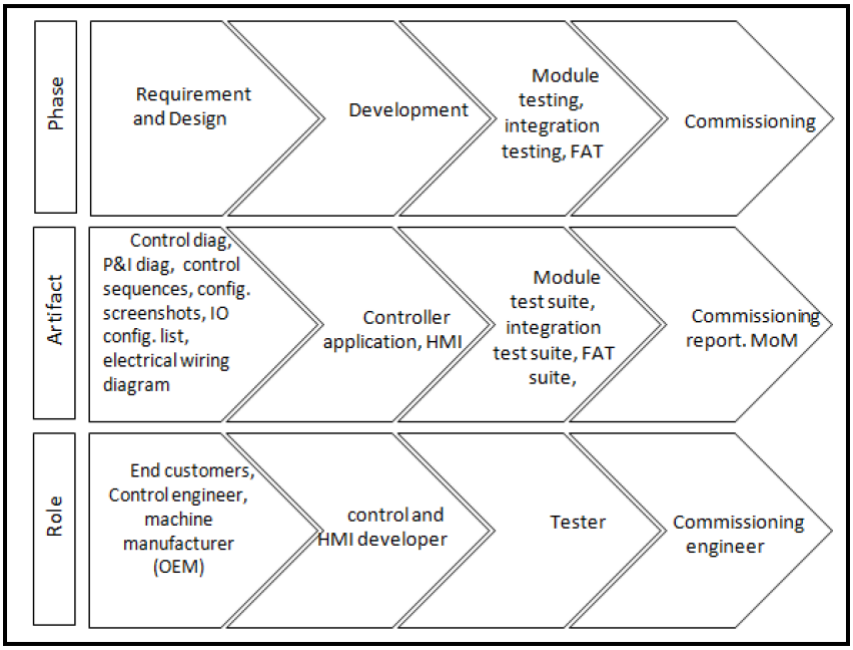
Testaus jakautuu neljään vaiheeseen, joita ovat moduulitestaus, integrointitestaus, tehdashyväksyntätesti (engl. Factory Acceptance Test, FAT) ja laitoshyväksyntätesti (engl. Site Acceptance Test, SAT). Moduulitestauksen tarkoituksena on yksittäisten moduulien toiminnallisuuksien validointi. IEEE:n [1] mukaan validointi on arviointiprosessi, jossa varmistetaan, että tuote täyttää asiakkaan tarpeet.

Dubeyn mukaan integrointitestauksessa testataan ohjelmistomoduulien väliset rajapinnat. IEEE:n [1] määritelmän mukaan integrointitestauksessa voidaan testata myös ohjelmisto- ja laitteistomoduulien välisiä vuorovaikutuksia.

Tehdashyväksyntätestissä (FAT) yhdistellään simulaattoreita reaalimaailman fyysiseen laitteistoon. FAT suoritetaan konfiguroimalla testattava systeemi, kokonaisuudessaan, kuten toimitettava laitos. Systeemin toimintosekvenssien validointi suoritetaan yhdessä asiakkaan kanssa. Jos asiakas hyväksyy testauksen kohteena olevan automaatioratkaisun, siirrytään SAT:iin. Standardin, IEC 61511-2 [29], mukaan tehdashyväksyntätestiä suositellaan, jos sovelluksen logiikka on kompleksinen tai turva-automaatiotoiminnoissa on redundanssijärjestelyjä. Redundanssijärjestelyillä pyritään lisäämään systeemin luotettavuutta ja vikasietoisuutta.

Dubeyn mukaan, laitoshyväksyntätesti suoritetaan käyttöönottovaiheen aikana. SAT perustuu asiakasvaatimuksiin. Käyttöönottovaiheen sovelluskehitys koostuu pääosin sovelluksen parametrien hienovirityksestä. Käyttöönottovaihe päättyy, kun systeemi on

stabiloitu ja asiakas on tyytyväinen käyttöönotettuun sovellukseen. Kuvassa 10 on esitetty automaatiosovelluksen kehityksen elinkaaren vaiheet sekä kuhunkin vaiheeseen liittyvät tuotokset ja vastuuroolit.




***Kuva 10.*** *Automaatiosovelluksen kehityksen elinkaari [28].*

Sekä ohjelmistotuotteen että fyysisen tuotteen elinkaarenhallinnan tavoitteena on alentaa tuotteen kustannuksia (kuten kehitys- ja ylläpitokustannukset). Elinkaarenhallinnan ja tuotekehityksen avulla pyritään myös tuottamaan arvoa niin asiakkaalle kuin tuottajallekin. Seuraavassa luvussa käsitellään virtuaalista tuotekehitystä ja sen yhteyttä elinkaarenhallintaan.

# 3. TUOTEKEHITYS JA TUOTETIETO

## 3.1 Simulointi osana tuotekehitystä

Simulointia voidaan hyödyntää kompleksisten systeemien tutkimuksessa ja kehityksessä. Näin saadaan tutkimustuloksia jo kehitysprosessin varhaisten vaiheiden aikana. Tässä luvussa tutustutaan mallipohjaiseen suunnitteluun ja siihen läheisesti liittyviin muihin mallinnuskonsepteihin. Lisäksi käsitellään mallipohjaisen suunnittelun hyödyntämistä mekatronisen systeemin virtuaalisessa tuotekehityksessä. Lopuksi käsitellään kehitettyjen ohjelmistojen testausta simulointityökalujen avulla.

### 3.1.1 Mallipohjainen suunnittelu

Mallipohjaisen suunnittelun ja kehityksen menetelmiin viitataan kirjallisuudessa useilla samankaltaisilla käsitteillä. Mallipohjaista suunnittelua voidaan soveltaa, sekä koko systeemin että ohjelmistojen kehitystyössä. Tämän luvun tarkoituksena on selventää eri mallinnuskonseptien välisiä yhteyksiä ja eroja. Mallinnuskonseptien määritelmät perustuvat ensisijaisesti lähteeseen [30], ellei erikseen toisin ilmaista.

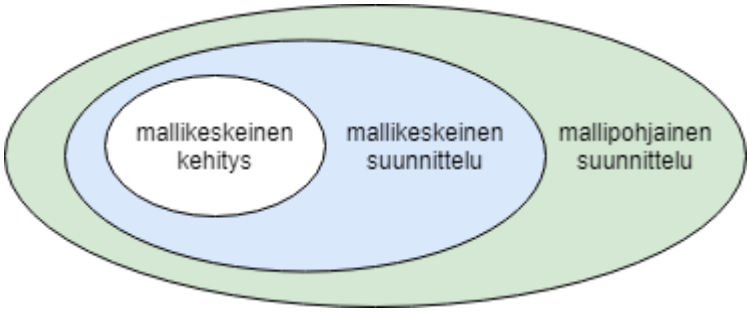
Mallipohjainen (engl. model-based) ja mallikeskeinen (engl. model-driven) lähestymistapa eroavat toisistaan mallien käyttötarkoituksen näkökulmasta. Mallipohjaisella suunnittelulla (engl. model-based engineering) tai mallipohjaisella kehityksellä (engl. model-based development) viitataan mallinnusprosessiin, jossa mallit eivät (välttämättä) ole kehitystyön kannalta ensisijaisia tuotoksia. Tällöin mallien käyttötarkoituksena on, mallimuunnoksien (engl. model transformation) sijaan, suunnitteluprosessin tuotosten dokumentointi ja verifiointi. Mallipohjaisella kehityksellä viitataan erityisesti ohjelmistojen kehittämiseen. Mallipohjainen suunnittelu sisältää ohjelmistokehityksen lisäksi systeemin analysoitiin liittyviä vaiheita. Mallipohjaiseen suunnitteluun viitataan kirjallisuudessa myös käsitteellä mallipohjainen systeemisuunnittelu (engl. model-based systems engineering) [31, 32]. MathWorks viittaa lähestymistapaan englanninkielisellä käsitteellä model-based design.

Mallikeskeisissä menetelmissä, kuten mallikeskeinen suunnittelu (engl. model-driven engineering) ja mallikeskeinen kehitys (engl. model-driven development), eksplisiittiset eli yksiselitteiset mallit ohjaavat suunnitteluprosessin jokaista vaihetta. Nämä menetelmät

perustuvat automaattisiin mallimuunnoksiin. Mallimuunnoksien avulla, alustariippumattomasta mallista voidaan generoida eli tuottaa alustakohtainen malli ja edelleen implementaatio (esim. lähdekoodi).

Mallikeskeisyyttä voidaan pitää mallipohjaisuuden alakäsitteenä. Mallikeskeisyys viittaa aktiviteetteihin, joissa mallit ovat suunnitteluprosessin keskeisimpiä tuotoksia. Mallipohjaiset menetelmät voivat siis sisältää myös mallikeskeisten menetelmien elementtejä, kuten automaattisen koodigeneroinnin.

Edellä kuvattujen määritelmien mukaisesti, mallikeskeinen kehitys voidaan ajatella mallikeskeisen suunnittelun osajoukkona ja mallikeskeisyys edelleen mallipohjaisuuden osajoukkona. Kuvassa 11 on havainnollistettu edellä kuvattujen käsitteiden välisiä yhteyksiä Venn-diagrammina.



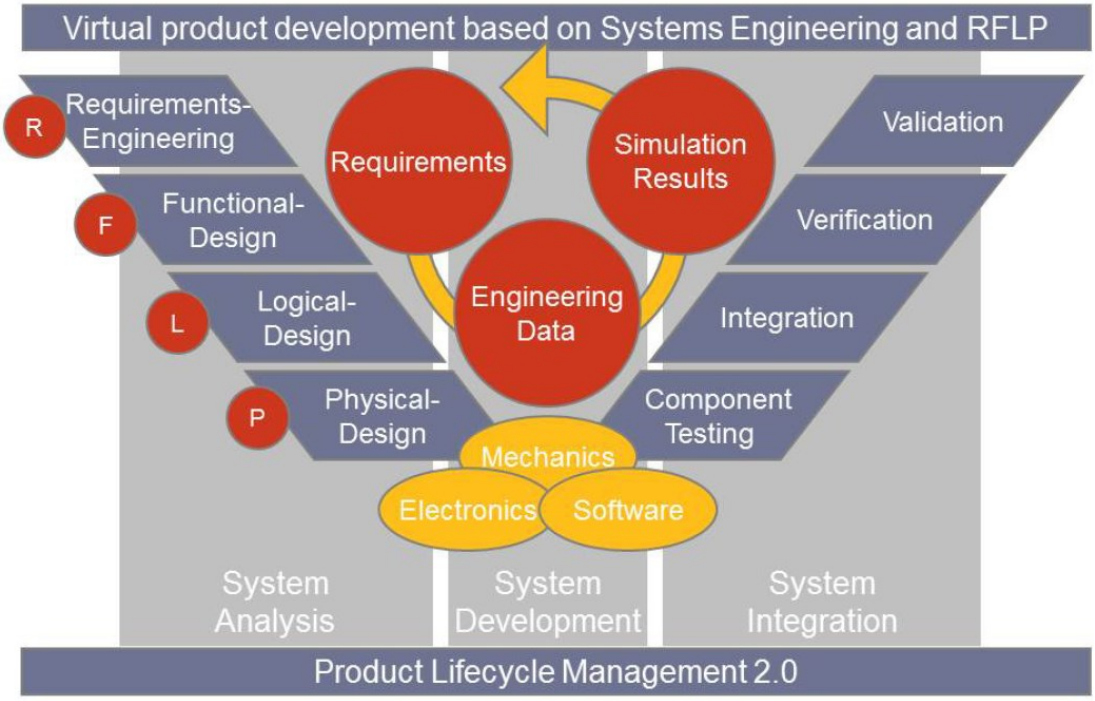

***Kuva 11.*** Mallinnuskonseptien väliset yhteydet, mukailtu lähteestä [30].

Tässä tutkimuksessa mallipohjaisella suunnittelulla tarkoitetaan suunnitteluprosessia, jossa hyödynnetään simulointityökaluja kompleksisten systeemien mallinnuksessa, kehityksessä, testauksessa ja validoinnissa. Erityisesti käsitellään suunnitteluprosessia, jonka tarkoituksena on kehittää ohjaussovelluksia mekatronisille systeemeille. Tällainen suunnitteluprosessin vaiheita ovat hallittavan systeemin mallinnus, säätimen suunnittelu ja analysointi, systeemin ja sitä ohjaavan säätimen simulointi sekä säätimen implementointi ja testaus. Suunnitteluprosessissa tuotettuja simulointimalleja voidaan käyttää esimerkiksi prototyyppien luomiseen kehitettävästä systeemistä sekä ohjelmistojen testaukseen ja validointiin. Seuraavassa alaluvussa esitellään eräs lähestymistapa mekatronisen systeemin mallipohjaiseen suunnitteluun.

### 3.1.2 Virtuaalinen tuotekehitys

Monet suunnitteluvirheet huomataan vasta käyttöönotto- tai käyttövaiheessa. Tällöin tuotannon käynnistyminen viivästyy. Virheiden tunnistaminen vasta tuotantolaitoksella lisää myös virheiden korjaukseen tarvittavaa työmäärää ja niihin liittyviä kustannuksia. Käyttöönotossa ilmenevien virheiden määrää voidaan vähentää testaamalla systeemiä jo tuotekehityksen aiemmissa vaiheissa [33].

Tässä luvussa esitetään mallipohjainen suunnitteluprosessi perustuen lähteeseen [32]. Mekatronisen systeemin tuotekehitys voidaan jakaa kolmeen päävaiheiseen, jotka ovat systeemin analysointi, kehitys ja integrointi. Tällainen systeemisuunnittelun prosessi voidaan esittää ns. V-mallin avulla. V-mallin vasen puoli kuvaa systeemin määrittelyä, keskiosa implementointia ja oikea puoli testausta. Mekatronisen systeemin tuotekehityksen V-mallia on havainnollistettu kuvassa 12.




***Kuva 12.*** *Mallipohjaiseen suunnitteluun perustuva virtuaalinen tuotekehitys [32].*

Mallipohjaisen suunnitteluprosessin ensimmäisenä vaiheena on vaatimusten analysointi. Vaiheen tarkoituksena on luoda vaatimusmalli, joka kuvaa kehitettävään tuotteeseen liittyvät asiakasvaatimukset. Seuraavissa vaiheissa nämä vaatimukset linkitetään tuotteen toiminnalliseen ja loogiseen rakenteeseen.

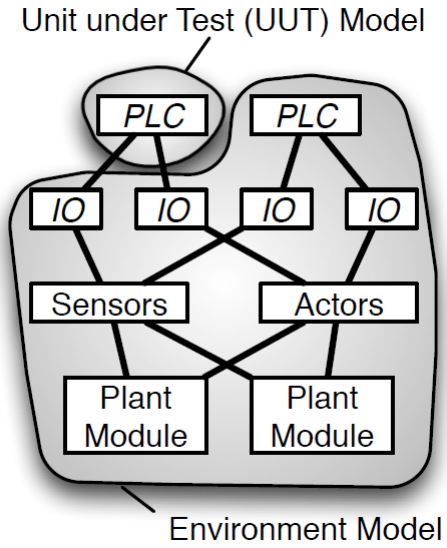
Toinen vaihe on toiminnallinen suunnittelu, jossa systeemille luodaan toiminnallinen rakenne. Systeemi jaetaan päätoimintoihin ja edelleen alitoimintoihin. Nämä toiminnot kuvataan lohkoina. Lohkojen väliset rajapinnat kuvataan energian, materiaalien ja informaation virtauksena. Näin voidaan muodostaa systeemin modulaarista rakennetta kuvaava toiminnallinen malli.

Loogisen suunnittelun tarkoituksena on kuvata systeemin dynaaminen käyttäytyminen. Kunkin moduulin toimintaperiaatteet määritellään niiden dynaamista käyttäytymistä kuvaavien mallien avulla. Loogisen suunnittelun lopputulos on simulointivalmis systeemi. Loogisen suunnittelun tuotosta kutsutaan loogiseksi malliksi.

Systeemin analysoinnin viimeisenä vaiheena on fysikaalinen suunnittelu. Tällöin loogista mallia täydennetään systeemin mekaanisia ominaisuuksia (kuten geometria ja kinematiikka) kuvaavien 3D CAD (engl. 3 Dimensional Computer Aided Design, 3D CAD) –mallien avulla. CAD-työkalulla luodut osat ja osakokoonpanot sisällytetään simulointimalliin. CAD-malleihin tehdyt muutokset vaikuttavat simuloinnin kohteena olevan systeemin käyttäytymiseen.
Systeemin analysointivaihetta seuraa kehitysvaihe. Kehitysvaiheessa simuloimalla voidaan arvioida ja optimoida systeemin suorituskykyä. Virtuaalinen tuotekehitys tukee myös mallipohjaisen ohjelmistokehityksen vaiheita, kuten koodin automaattista generointia ja mallipohjaista testausta.
Kehitysvaiheen jälkeen siirrytään integrointivaiheeseen. Tällöin testataan moduulit ja niiden väliset rajapinnat. Verifiointi suoritetaan vertaamalla testitapausten simulointituloksia vaatimusten analysointi –vaiheessa tunnistettuihin asiakastarpeisiin., V-mallin oikea puoli vastaa valmistajan sisäisen testauksen, moduulitestauksen ja integrointitestauksen, vaiheita. Sisäisen testauksen jälkeen suoritetaan tehdashyväksyntätesti (FAT) yhdessä loppuasiakkaan tai laitosintegraattorin kanssa.

### 3.1.3 Hardware in the loop (HIL) –simulointi

HIL-simulointi (engl. Hardware In the Loop simulation) on yksi reaaliaikasysteemien kehitykseen ja testaukseen käytetyistä menetelmistä, jota voidaan käyttää sekä sisäisessä testauksessa että osana tehdashyväksyntätestiä (FAT). Tässä luvussa HIL-simulointi esitetään ensisijaisesti lähteen [33] mukaan, ellei toisin mainita. HIL-simulointi on lähestymistapa, jolla testataan fyysistä laitteistoa (kuten logiikkaohjaimia). Tällöin ohjausso-vellus ladataan oikeaan ohjainlaitteeseen (esim. PLC). Sovelluksen toimintaa siis testataan, fyysisellä ohjainalustalla, virtuaalisessa simulointiympäristössä [34].
Systeemiä, johon testauksen alainen ohjauslaitteisto liitetään, simuloidaan siten että testattava laitteisto luulee olevansa liitettynä reaalimaailman systeemiin. Modulaariset simulointimallit mahdollistavat, testauksen alaisen, fyysisen laitteiston poistamisen simulointiympäristön mallista. Modulaarisuus siis nopeuttaa testiympäristön konfigurointia. Kuvassa 13 on havainnollistettu HIL-simuloinnin periaatetta.




***Kuva 13.*** *HIL-simuloinnin testausjärjestely [33].*

Testimallilla kuvataan testauksen alaista laitteistoa. Testimallin käyttötarkoituksena on testitapausten tuottaminen fyysiselle laitteistolle. Logiikkaohjainta (PLC) testattaessa, anturien mittausarvoja kuvaavat herätesignaalit lähetetään ohjainyksikölle. Tämän jälkeen, fyysisen ohjainyksikön tuottamia ohjaussignaaleja verrataan testimallin ennustamiin ohjauksiin.

HIL-testauksessa voidaan käyttää joko yhden mallin tai kahden mallin lähestymistapaa. Yhden mallin lähestymistavassa sekä testitapausten generointiin että lähdekoodin generointiin käytetään samaa mallia. Koska implementaatio ja testitapaukset ovat peräisin samasta mallista, HIL-simuloinnilla voidaan testata vain generoidun koodin suorituskykyä suhteessa reaaliaikavaatimuksiin.

Kahden mallin lähestymistavassa testimalli ja ohjaussovelluksen koodin implementointi toteutetaan toisistaan erillään, esimerkiksi ohjelmoimalla lähdekoodi manuaalisesti. Tällöin HIL-simuloinnilla voidaan testata suorituskyvyn lisäksi myös toiminnallisia ominaisuuksia. Kahden mallin (testimalli ja ohjelmointimalli) tuottama redundanttinen informaatio parantaa testauksen luotettavuutta. Redundanssi kuitenkin lisää myös testauksen kustannuksia.

Isermannin (et al.) [35] mukaan HIL-simuloinnin etuja ovat:

- kehitysajan lyheneminen ja kustannussäästöt
- ohjauslaitteiston ja ohjelmiston testaaminen vaarallisissa sekä vaativissa käyttöolosuhteissa
- systeemin vikojen ja niiden vaikutusten testaaminen
- testien uusittavuus ja toistettavuus.

Simuloinnilla jäljitellään reaalimaailman systeemin käyttäytymistä. Simulointi voidaan suorittaa vain malliin pohjautuen. Simulaation laatiminen siis edellyttää mallintamista.

## 3.2 Mallinnus

Tässä luvussa käsitellään mallien käyttötarkoituksia sekä mallinnusmenetelmiä ja -kieliä. Erityisesti tarkastellaan systeemin hierarkkisen rakenteen mallintamista, mikä mahdollistaa systeemin modularisoinnin. Luvun tarkoituksena on havainnollistaa, miten hierarkkisuus tukee simulointimallien ja ohjelmistojen uudelleenkäytettävyyttä.

### 3.2.1 Mallien käyttötarkoitukset

Mallit ovat systeemien abstraktioita, joiden tarkoituksena on kuvata systeemin oleelliset ominaisuudet [31]. Mallinnuksen näkökulmana voi olla esimerkiksi systeemin vaatimukset, rakenne, toiminnot tai dynaaminen käyttäytyminen [36]. Nämä näkökulmat vastaavat, kuvan 12 mukaisen, V-mallin vasenta puolta.

Malli voidaan esittää esimerkiksi tekstinä ja matriisina tai graafina [36]. Mallien esittämiseksi on kehitetty lukuisia standardoituja mallinnuskieliä.

Niggemann & Stroop [37] jaottelevat mallit, käyttötarkoituksiensa perusteella, toimintojen ja systeemien kehitykseen käytettäviin malleihin. Selvyyden vuoksi, toimintojen kehitykseen käytettäviä malleja kutsutaan loogisiksi malleiksi ja systeemien kehitykseen käytettäviä malleja systeemimalleiksi. Kokonaisvaltaisen systeemikuvauksen luomiseksi tarvitaan sekä systeemimalleja että loogisia malleja. Systeemimallien ja loogisten mallien yhdistäminen mahdollistaa luvussa 3.1.2 esitellyn mallipohjaisen suunnitteluprosessin.

Systeemimallit määrittelevät systeemin vaatimukset, hierarkkista rakenteen ja moduulien väliset rajapinnat. Niggemannin & Stroopin [37] mukaan nämä mallit on tarkoitettu ensisijaisesti ihmisten käytettäviksi, esimerkiksi visualisointi- ja dokumentointitarkoituksiin. Kompleksinen systeemi on helpompi ymmärtää, kun visualisointiin käytetään yksinkertaistettuja abstraktioita. Dokumentoituja systeemimalleja voidaan käyttää kehitysprojektiin osallistuvien osapuolien välisinä sopimuksina. Tällainen sopimus voi olla esimerkiksi rajapintamäärittely.

Loogiset mallit kuvaavat ohjausalgoritmeilla tai matemaattisilla yhtälöillä tietyn moduulin käyttäytymisen [38]. Loogisia malleja voidaan käyttää dokumentoinnin ja visualisoinnin lisäksi simulointiin ja automaattiseen koodin generointiin. Koodigenerointityökalulla voidaan vähentää manuaalisen ohjelmointityön määrää. Simulointi mahdollistaa ohjelmointivirheiden löytämisen ja implementoitujen toimintojen verifioinnin jo kehitysprosessin alkuvaiheessa.

Mallit ovat yksinkertaistuksia reaalimaailman systeemistä. Mikään malli ei täysin vastaa todellisuutta, vaan on aina approksimaatio. Mallinnettaessa on siis päätettävä yksityis-

kohtien määrä kullakin mallinnustasoilla. Näitä mallinnustasoja kutsutaan abstraktiotasoiksi. Seuraavassa alaluvussa esitellään abstraktiotasoihin perustuvia hierarkkisia mallinnustapoja.

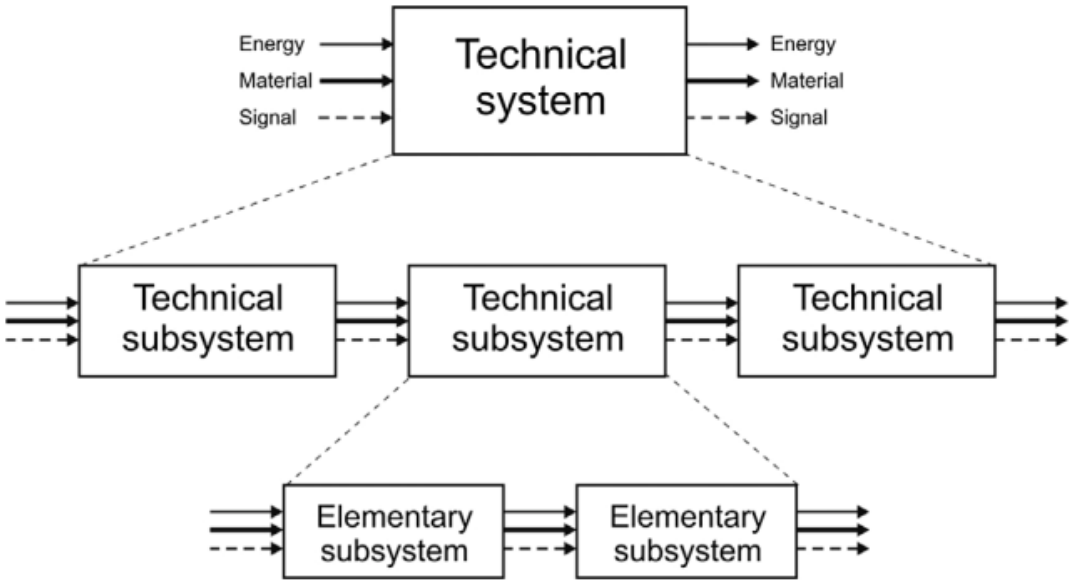
### 3.2.2 Hierarkkinen systeemimalli

Tässä luvussa mallinnustavat esitetään ensisijaisesti Hehenbergerin [39] ja Niggemannin [40] tutkimuksiin perustuen, ellei toisin ilmaista. Perinteisesti mekaanisen osasysteemin suunnittelu on ollut mekatronisen systeemin suunnittelun vaikein ja kompleksisin osuus [41]. Ohjelmistolla toteutettavien toiminnallisuuksien määrä on kuitenkin kasvanut huomattavasti teknologisen kehityksen myötä [21]. Kehittyneissä laitteissa, myös liikkeenohjaukseen käytettävät algoritmit ovat entistä kompleksisimpia. Niggemannin mukaan, kompleksisuudesta aiheutuvia ongelmia pyritään ratkaisemaan mallipohjaisella suunnittelulla ja modulaarisella ohjelmoinnilla sekä alustan standardoinnilla.

Hehenberger esittelee hierarkkisen mallinnustavan, jonka avulla pelkistetyistä eli redusoiduista malleista voidaan siirtyä kohti yksityiskohtaisia malleja. Korkean abstraktiotason redusoituja malleja voidaan käyttää kokonaissysteemin optimointiin. Matalampien abstraktiotasojen yksityiskohtaisia malleja, sen sijaan, käytetään yksittäisten toimintojen kehittämiseen. Yksityiskohtien määrä siis kasvaa siirryttäessä korkeammalta abstraktiotasolta matalampaan. Abstraktiotasoilla voidaan siis lisätä mallien uudelleenkäytettävyyttä ja yleispätevyyttä.

Systeemimalli voidaan jakaa pienempiin ja yksityiskohtaisempiin osamalleihin hyödyntäen abstraktiotasoja. Abstraktio on prosessi, jossa systeemimallia yksinkertaistetaan kuvaamalla vain sen tärkeimmät ominaispiirteet, joten abstraktio määrittelee systeemin abstraktien olioiden ja niiden ominaisuuksien avulla sekä olioiden väliset kommunikointitavat eli rajapinnat.

Hehenbergerin mukaan tärkeitä ominaisuuksia tekniselle systeemille ovat toiminnot, rakenne ja käyttäytyminen. Kompleksinen systeemi koostuu useista toiminnoista, jotka muodostavat hierarkkisen rakenteen. Abstraktiotasojen avulla, kompleksinen systeemi jaetaan vähemmän kompleksisiin alisysteemeihin eli moduuleihin. Moduulien väliset riippuvuudet pyritään minimoimaan ja hallitsemaan hyvin määritellyillä rajapinnoilla. Kuvassa 14 on esitetty teknisen systeemin hierarkkinen toiminnallinen erittely, jossa rajapintoja kuvataan energian, materiaalien ja signaalien (informaation) virtauksena.




***Kuva 14.*** *Systeemin toiminnallinen erittely [39].*

Kuten kuvasta 14 nähdään, Hehenberger ehdottaa systeemin mallintamista kolmeen abstraktiotasoon. Nämä abstraktiotasot ovat:

- korkean tason systeemimalli
- pelkistetyt osasysteemien mallit
- osasysteemin komponenttien yksityiskohtaiset mallit.

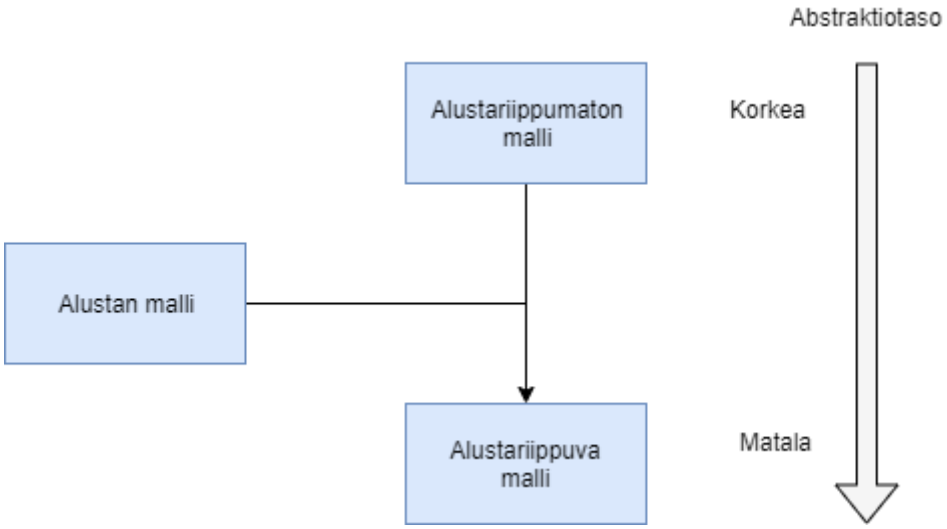
Korkean tason systeemimalli soveltuu dokumentointiin ja visualisointiin. Simulaattorin yleispätevä rakenne saadaan jakamalla kokonaissysteemi pelkistettyihin osasysteemeihin. Yksityiskohtaiset mallit mahdollistavat 3D-simuloinnin ja yksittäisten toimintojen, ohjelmiston ominaisuuksien, mallipohjaisen kehittämisen.

Niggemann esittelee erityisesti automaatio-ohjelmistojen ja simulointimallien uudelleenkäytettävyyttä tukevan mallinnustavan. Tämä mallinnustapa perustuu signaalien informaatiosisältöä ja merkitystä eli semantiikkaa kuvaavaan abstraktiotasoon. Mallinnusmenetelmä pohjautuu Object Management Groupin kehittämään mallikeskeiseen arkkitehtuuriin (engl. Model Driven Architecture, MDA). MDA keskittyy ohjelmistojen suunnitteluun, kehittämiseen ja implementointiin. MDA-malleja käytetään muun muassa ohjelmistojen määrittelyjen jäsentelyyn.

Alustariippumattomien mallien (engl. Platform Independent Model) abstraktiotaso on alustariippuvien mallien (engl. Platform Specific Model) abstraktiotasoa korkeampi. Alustariippumaton malli koostuu ohjelmistokomponenteista ja prosessimalleista. Prosessien mallintamiseen voidaan käyttää esimerkiksi MATLABin Simulink-työkalua. Ohjelmistokomponentit ovat esimerkiksi säätöalgoritmeja tai diagnostiikka-algoritmeja. Näiden ohjelmistokomponenttien välisien rajapintojen abstraktiotasona käytetään prosessimallien määrittelemiä signaaleja eli prosessisignaaleja.

Ohjelmistojen uudelleenkäytettävyyttä voidaan lisätä valitsemalla prosessisignaalit reaalimaailman prosessin fyysisten tilojen mukaisesti. Prosessisignaalit ovat antureiden ja toimilaitteiden arvoja kuvaavia muuttujia. Fysikaaliset rajapinnat ovat helpommin ymmärrettävissä ja ylläpidettävissä kuin mielivaltaisiin sopimuksiin perustuvat rajapinnat. Tällöin ohjelmistomoduulien rajapintoja ei tarvitse sopia ohjelmistokohtaisesti, vaan rajapinnat määräytyvät reaalimaailman laitoskonfiguraatioon perustuen.

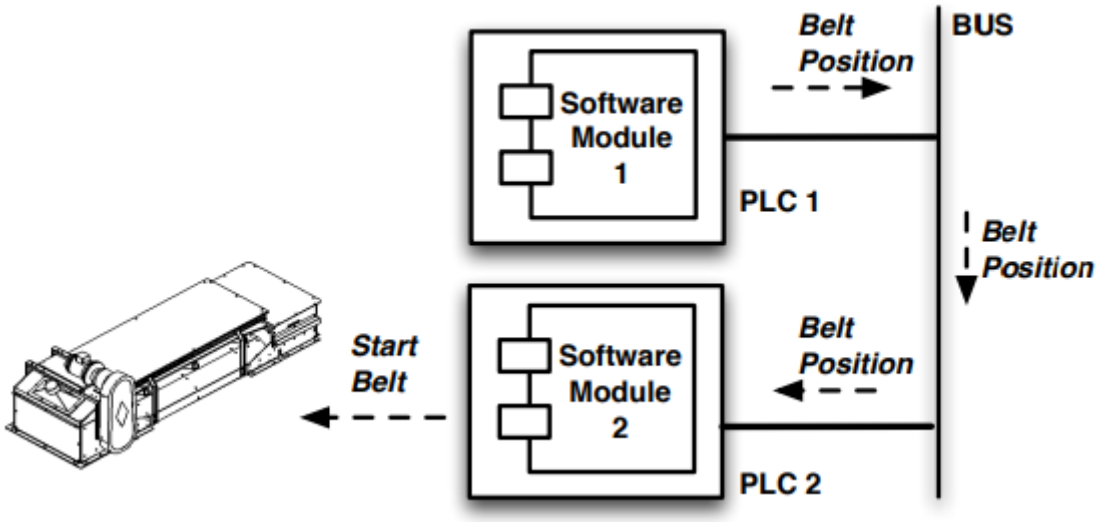
Alustan malli sisältää laitteiston (engl. hardware) eli logiikkaohjaimet (PLC), kommunikaatioväylät ja IO-laitteet (engl. Input and Output, I/O). Alustariippuva malli saadaan yhdistämällä alustariippumaton malli ja alustan malli. Menettely on esitetty kuvassa 15.




***Kuva 15.*** *MDA-pohjaisen mallinnuksen hierarkiatasot, mukailtu lähteestä [40].*

Alustariippuvassa mallissa prosessisignaalit kohdistetaan väyläsignaaleihin ja I/O-laitteiden portteihin. I/O-laiteet liittävät PLC-laitteen kenttälaitteisiin (kuten moottori tai anturi). Lisäksi ohjelmistomoduulit kohdistetaan tietylle PLC-laitteelle. Alustariippuva malli siis kuvaa tietyn laitteiston ja ohjelmiston välisiä yhteyksiä.

Mallinnuksen näkökulmasta prosessit, ohjelmistomoduulit ja väylät kuvataan ns. black box –olioina, jotka tarjoavat ja vaativat prosessisignaaleja. Prosessisignaalien attribuutteina ovat signaalin nimi, tyyppi (kuten analoginen tai digitaalinen) ja arvoväli eli minimi- ja maksimiarvo. Black box –kuvauksessa sisään- ja ulostuloja eli rajapintoja voidaan tarkastella ilman tietoa mallin sisäisestä rakenteesta tai käyttäytymisestä. Back box –lähestymistapa tukee siis systeemin jakamista moduuleihin. Kuvassa 16 on esitetty hihnakuljettimen liikkeenohjausta kuvaava alustariippuva malli.




***Kuva 16.*** *Alustariippuva malli hihnakuljettimelle [40].*

### 3.2.3 Mallinnuskielet ja tiedonvaihto

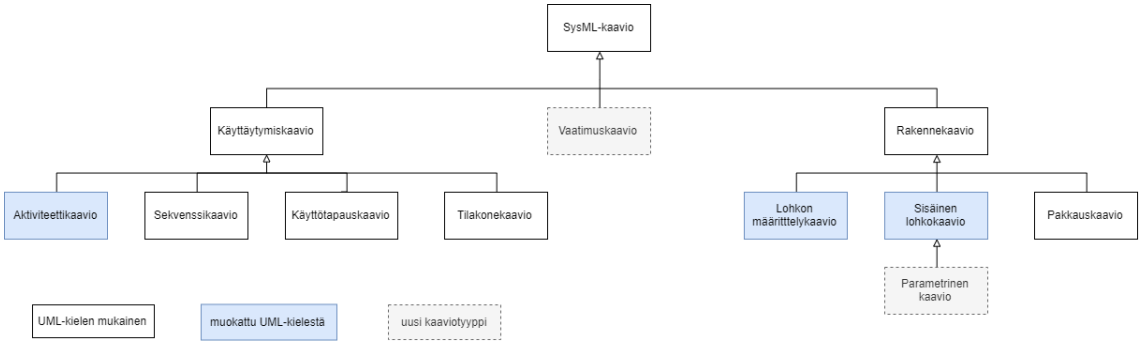
Lukuisten standardoitujen mallinnuskielien joukosta esitellään, laajalti käytetyt, Unified Modeling Language (UML) ja Systems Modeling Language (SysML). Lisäksi tutustutaan tiedonvaihtoon ja -tallennukseen kehitettyyn Automation Markup Language (AutomationML) –merkintäkieleen. Näiden mallinnus- ja merkkauskielien vahvuuksia ja heikkouksia tuodaan esiin systeemisuunnittelun ja hierarkkisen mallintamisen näkökulmista.

Mallinnuskielien käyttökohteita ovat esimerkiksi systeemi-, ohjelmisto- ja laitesuunnittelu. Mallinnus vie tavallisesti paljon aikaa. Mallinnuskielen ja esitystavan valinta riippuu muun muassa tarvittavien yksityiskohtien ja näkökulmien määrästä sekä mallien käyttötarkoituksista. Mallinnuskielen valintaan tulee siis kiinnittää huomiota.

Reichwein & Paredisin [31] mukaan malleja voidaan kuvailla kolmen kriteerin avulla, joita ovat tarkkuus, täsmällisyys ja monitulkintaisuus. Tarkkuus liittyy mallin oikeellisuuteen ja täsmällisyys yksityiskohtaisuuteen. Monitulkintaisuus on ominaisuus, joka pyritään minimoimaan. Standardoitujen mallinnuskielien symbolit ovat ennalta määriteltyjä, mikä vähentää mallien tulkinnanvaraisuutta.

Unified Modeling Language (UML) on standardoitu ja laajalti tunnettu mallinnuskieli. UML on graafinen mallinnuskieli, jonka alkuperäisenä käyttötarkoituksena on ollut oliopohjaisen ohjelmistokehityksen tukeminen. UML-kieli kuitenkin soveltuu myös mallipohjaisen suunnittelun ja liiketoimintaprosessien mallinnustarpeisiin. Kielen vahvuuksina voidaan pitää kattavaa työkalutukea sekä erilaisten kaavioiden ja näkökulmien tarjoamaa valinnanvaraa [31]. Yleiskäyttöön suunnattu ja laaja mallinnuskieli ei välttämättä ole paras mahdollinen vaihtoehto tukemaan yksityiskohtaista systeemien tai automaatiotoimintojen mallintamista. UML kuitenkin mahdollistaa alakohtaisten mallien luomisen profiileja hyödyntämällä. Profiilit siis tarjoavat mekanismin UML-mallien laajentamiseen.

Systems Modeling Language (SysML) on jalostettu systeemisuunnittelun tarpeisiin UML-kielen pohjalta. SysML-kielestä on karsittu pois monia UML-kielen ohjelmistokeskeisiä kaaviotyyppejä. Lisäksi SysML määrittelee uusia kaaviotyyppejä (profiileja), kuten vaatimuskaavio ja parametrinen kaavio. SysML soveltuu siis UML-mallinnuskieltä paremmin vaatimustenhallintaan ja rajoitteiden määrittelyyn. SysML on myös UML-kieltä yksinkertaisempi. UML-kielessä on 13 erilaista kaaviotyyppiä, kun taas SysML-kielessä niitä on 9. SysML on siis UML-kieltä helpompi oppia ja käyttää. SysML-kielen kaaviotyypit ja niiden yhteydet UML-kaavioihin on esitetty kuvassa 17.

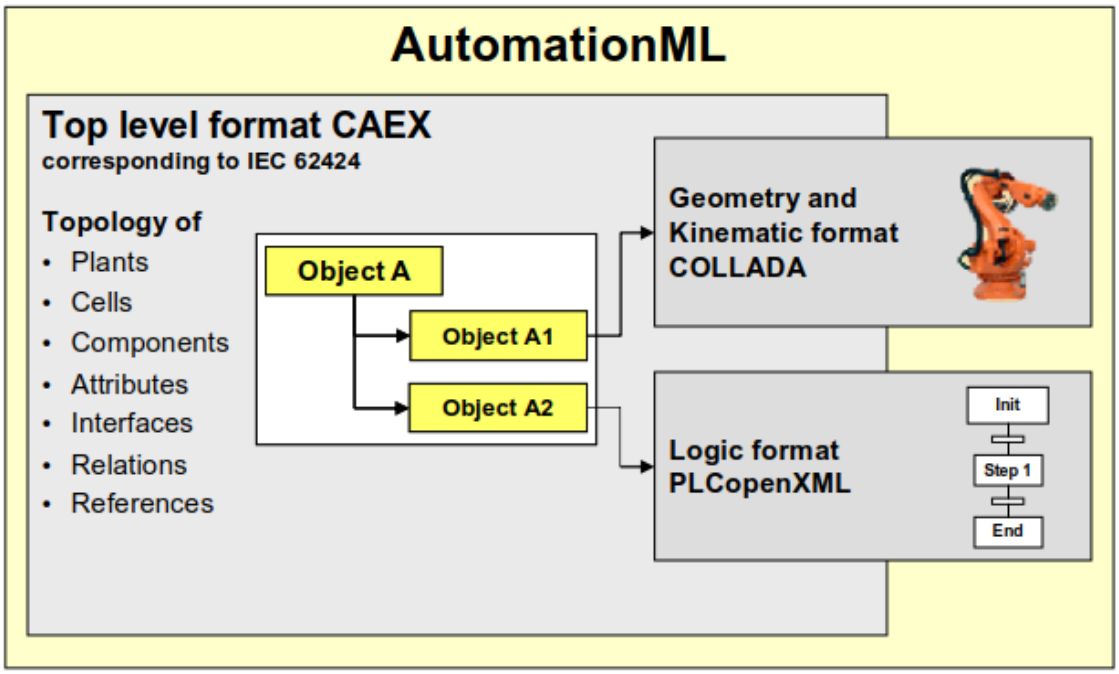



***Kuva 17.*** *SysML-kaaviotyypit.*

Automation Markup Language (AutomationML) on XML-merkintäkieleen (engl. Extensible Markup Language, XML) pohjautuva tietoformaatti (engl. data format). Avoimen ja ilmaisen AutomationML-standardin tavoitteena on erilaisten, valmistajakohtaisten, suunnittelutyökalujen välisen tiedonvaihdon tukeminen. Mekatronisen systeemin suunnittelussa päällekkäistä ja uudelleenhyödynnettävää informaatiota voidaan tuottaa esimerkiksi mekaniikkasuunnittelun, sähkösuunnittelun ja PLC-ohjelmoinnin työkaluilla. AutomationML-standardin kehitykseen on osallistunut sekä akateemisia tutkimuslaitoksia että teollisuusyrityksiä (kuten Siemens, ABB ja Rockwell). Monet teollisuusautomaatiossa käytetyt suunnittelutyökalut hyödyntävät AutomationML-standardia.

AutomationML tallentaa systeemin hierarkkisen rakenteen CAEX-standardin mukaisessa tietoformaatissa ja mahdollistaa siten systeemin rakenteen sekä siihen liittyvien komponenttien mallintamisen oliopohjaisilla abstraktioilla. Luokkien ilmentymille eli olioille voidaan määritellä yhtenäiset ja yksilölliset ominaisuudet attribuutteina. Olioiden väliset yhteydet voidaan esittää instanssihierarkian ja rajapintakirjastojen (engl. interface class library) avulla. Roolikirjastot (engl. role class library) mahdollistavat olioiden jaottelun abstrakteihin, valmistajariippumattomiin, komponenttiluokkiin (kuten moottori, anturi tai PLC). Valmistajakohtaiset tuotteet (kuten Siemens S7-300 –logiikkaohjain) voidaan jäsennellä systeemiyksikkökirjastojen (engl. system unit class library) luokkarakenteisiin. CAEX-standardin lisäksi AutomationML tukee COLLADA-standardia, jolla voidaan tallentaa 3D CAD –mallien geometria- ja kinematiikkatiedot. Myös olioiden käyttäytymiskuvaukset voidaan tallentaa PLCopen XML-standardin mukaisina toimintosekvensseinä.

Sekä CAD-mallien että ohjaussovelluksen kuvaukset voidaan liittää osaksi systeemin integroitua tietomallia. AutomationML-tietomallin rakennetta on havainnollistettu kuvassa 18.




***Kuva 18.*** *AutomationML-formaatin rakenne [42].*

CAEX, COLLADA ja PLCopen XML -standardien ohella, AutomationML-tietoformaattia voidaan laajentaa millä tahansa XML-pohjaisella formaatilla. Esimerkiksi, systeemin komponenttien fysikaaliset käyttäytymiskuvaukset voidaan liittää AutomationML-tietomalliin simuloitavina FMU-moduuleina (engl. Functional Mockup Unit, FMU) [38]. FMU on monien mallinnustyökalujen, kuten Mathworksin Simulinkin, tukema rajapintastandardi simulointimallien jakamiseen sidosryhmien ja eri valmistajien työkalujen välillä. FMU-rajapinnat perustuvat black box –malleihin ja niiden rajapintojen määrittelemiseen matemaattisten yhtälöiden avulla.

AutomationML-kielen vahvuusalueena on siis alakohtainen ja yksityiskohtainen suunnittelu sekä niihin liittyvän informaation mallintaminen. SysML-kieli soveltuu paremmin korkean tason systeemisuunnitteluun. UML:n vahvuutena on ohjelmistojen korkean tason suunnittelu. Lisäksi SysML ja UML ovat visuaalisesti ilmaisuvoimaisempia kieliä verrattuna AutomationML-kieleen.

Mallipohjainen suunnittelu perustuu kokonaisvaltaiseen systeeminmalliin. Mallinnusmenetelmän ja mallinnuskielen lisäksi tarvitaan myös mallinnustyökalu. Mallipohjaisen suunnittelun työkaluilla, kuten MathWorksin Simulink voidaan luoda dokumentteja mallien pohjalta. Tällöin suunnitteluprosessin tuotoksia on mahdollista hallita dokumenttipohjaisesti.

## 3.3 Tuotetiedon hallinta

Tuotetiedon hallinnalla (engl. Product Data Management, PDM) viitataan tavanomaisesti aineellisiin hyödykkeisiin liittyvään tuotetietoon. PDM-työkalua käytetään tuotetietoon kohdistuvien muutosten seuraamiseen ja julkaisumenettelyiden implementointiin. Ohjelmistotuotannossa, vastaaviin aktiviteetteihin viitataan käsitteellä versionhallinta. Tässä luvussa tutustutaan ohjelmistoihin ja mekatronisiin systeemeihin liittyvään tuotetietoon ja käsitellään tuotetiedon hallintaan kehitettyjen työkalujen soveltuvuutta ohjelmistojen versionhallintaan.

### 3.3.1 Ohjelmistokehityksen dokumenttilajit

Versionhallintatyökalusta riippumatta, versionhallintaan vietävät dokumenttilajit ja niitä vastaavat dokumentit (kuva 2) tulee määritellä. Attribuuteilla voidaan yksilöidä nämä dokumentit. Attribuuttien eli metatietojen tallennusmahdollisuudet saattavat kuitenkin vaihdella työkalukohtaisesti. Tässä luvussa määritellään ohjelmistokehitysprojektiin liittyvät dokumenttilajit sekä dokumenttien attribuutit alan standardeihin ja kirjallisuuteen perustuen.

Standardi IEC 62708 [43] käsittelee sähkö- ja automaatioprojektien dokumenttilajeja sekä niiden sisältöä. Standardin dokumenttilajeista, PLC-ohjelmistojen kehityksen kannalta, oleellisimpia ovat:

- putkitus ja automaatiokaavio
- sähkö- ja automaatioprosessin liitäntöjen määrittely
- instrumenttidatalehti
- ohjausjärjestelmän rakennekaavio
- toimintokuvaus
- toimintojen lohkokaavio
- signaaliluettelo
- tulo/lähtö-luettelo
- konfigurointiparametriluettelo
- kynnysarvoluettelo.

Laitteiden turvallisuutta käsittelevät standardit ISO 13849-1 [44] ja IEC 62061 [45]. ISO 13849-1 esittelee suoritustasot (engl. performance level, PL) ja riskitasot. IEC 62061 kuvailee turvallisuuden eheyden tasoja (engl. safety integrity level, SIL). SIL-luokitusten ja PL-luokitusten määrittelyt sekä riskiarviot ovat tärkeä osa turva-automaatiotoimintojen kehitystä.

Walkerin (et al.) [15] mukaan mallipohjaisen suunnitteluprosessin dokumentoitavia tuotoksia ovat esimerkiksi simulointitulokset, testitapaukset ja testikattavuusraportit sekä ohjaussovelluksen generoitu koodi. Lisäksi versionhallintaan viedään suunnitteluprosessissa tuotetut simulointimallit ja niihin liittyvät parametrien konfigurointitiedostot.
Ohjelmistojen versiointiin liittyviä attribuutteja ovat versionumero ja versiotagit, viimeisimmän muutoksen ajankohta ja muutoksen tekijä sekä muutokseen liittyvä kommentti [46]. Versiotageilla voidaan merkitä esimerkiksi kantaversiot. Kommenteilla taas on tarkoitus kertoa, mitä on muutettu ja miksi.
Versiointiin voidaan käyttää, numeroiden lisäksi myös kirjaimia. Tämä on tyypillinen käytäntö tuotetiedon hallintaan tarkoitetuissa työkaluissa. PDM-työkalun versiointitunnukset voivat olla, esimerkiksi, muotoa A.1, A.2 tai B.1. Seuraavassa alaluvussa tutustutaan tarkemmin PDM-työkalujen tapaan hallita tuotetietoa ja tuotteiden eri versioita.

### 3.3.2 Tuotetieto ja PDM

Tuotetieto kuvaa tuotteen fyysisiä ja toiminnallisia ominaisuuksia. Nämä tiedot karakterisoivat tietyn tuotteen esimerkiksi kuvien ja mallien avulla. Tyypillistä tuotetietoa ovat erilaiset dokumentit, kuten tekniset kuvaukset, CAD-piirustukset ja 3D-mallit sekä piirikaaviot. Tuotetietoa voivat olla myös tuotteen markkinointiin tai käyttövaiheeseen liittyvät dokumentit. Tällaisia, asiakkaalle suunnattuja, dokumentteja ovat esimerkiksi käyttö- ja huolto-ohjeet sekä esitteet ja hinnastot.
PDM-työkalua voidaan käyttää myös ohjelmistojen versionhallintaan [22, 47, 48]. Suurin puute tekstitiedostopohjaisen lähdekoodin hallinnassa on sulauttamistoiminnallisuuden (engl. merging functionality) puuttuminen [47,48]. Kuten luvussa 2.2.3 todettiin, tässä tutkimuksessa käsitellään pääosin binääritiedostojen versionhallintaa. Sulauttamistoiminnallisuuden puuttuminen ei siis ole este PDM-työkalun käyttämiseen ohjelmistojen versionhallinnassa.
PDM-työkaluissa tiedonhallinta perustuu nimikkeisiin. Sääksvuoren & Immosen [22] mukaan nimikkeet ovat hierakkisia rakenteita, jotka jaotellaan loogisesti luokkiin. Nimikkeet toimivat systemaattisena ja standardoituna tapana tunnistaa ja nimetä tuote. Nimikkeet numeroidaan ja muodostetaan näin nimikehierarkia. Nimikkeillä voidaan hallita myös ohjelmistoja, kuvaamalla ohjelmiston toiminnot ja ominaisuudet nimikkeinä.
PDM-työkalulle tyypillisiä ominaisuuksia ovat: [14,22]

- nimikkeiden hallinta
- tuoterakenteen hallinta ja ylläpito
- tiedostojen ja dokumenttien hallinta

- dokumenttien ja nimikkeiden tilakirjanpito
- historiakirjaus
- muutostenhallinta
- konfiguraationhallinta
- käyttöoikeuksien hallinta
- informaationhaku
- työnkulun ja tehtävien sekä viestien (kuten sähköposti) hallinta
- varmuuskopiointi ja alkuperäisen kopion (engl. master copy) suojaus
- tiedostojen varastointi.

Työkalu varastoi tiedostot paikallisverkossa sijaitsevalle keskitetylle palvelimelle [22]. Tiedostojen hallinta perustuu palvelimelle tallennetun tiedon indeksointiin. Keskitetty palvelin ylläpitää oikeita versiointimenetelmiä, käyttöoikeuksia ja nimikkeisiin liittyviä metatietoja. Bricognen (et al.) [26] mukaan PDM-työkalun työnkulun hallinta –ominaisuus mahdollistaa myös (kuvan 3 mukaisen) muutostenhallintaprosessin. Edellä kuvattujen ominaisuuksien perusteella voidaan olettaa PDM-työkalun soveltuvan sekä systeemien konfiguraationhallintaan että ohjelmistojen versionhallintaan.

Budan (et al.) [14] mukaan PDM-työkalun toiminnallisuudet mahdollistavat myös simulointimallien versionhallinnan. Haasteena kuitenkin on, ettei simulointimalleja ja niihin liittyvää tietoa voida kuvata yksinkertaisella tuoterakenteella. Tämä johtuu siitä, että simulointitapauksia ja -malleja varioidaan sekä ketjutetaan toistensa kanssa. Simulointitiedot ja niiden väliset yhteydet muodostavat siis kompleksisen verkoston.

Mekatronisen systeemin tuotetiedon hallinta PDM-työkalulla vaatii erilaisten alakohtaisten tuotetietojen integrointia. Mekatronisen systeemin tuotetiedon (mekaniikka- ja sähköautomaatiolaitteisto sekä ohjelmisto) hallintaa varten on valittava siihen soveltuva tuoterakenne. Seuraavassa alaluvussa tutustutaan kirjallisuuden esittelemiin ratkaisumalleihin.

### 3.3.3 Mekatronisen systeemin tuotetiedon hallinta

Tämän luvun tarkoituksena on esitellä lähestymistapoja, jotka mahdollistavat mekatroniseen systeemiin liittyvän tuotetiedon integroinnin yhteiseen tietokantaan. Tuotetiedon integrointi mahdollistaa, luvun 3.1.2 mukaisen, virtuaalisen tuotekehityksen. Integraatio tukee myös mallipohjaista sovelluskehitystä.

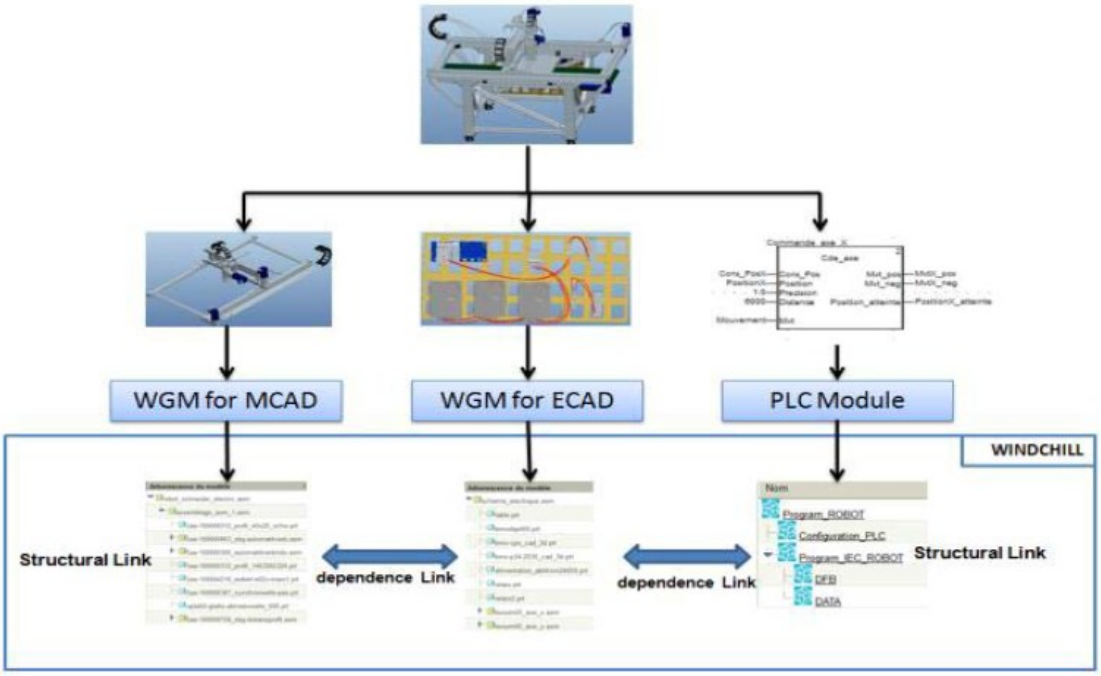
Ohjelmistojen versionhallinta perustuu tyypillisesti kokonaisten tiedostojen ja rinnakkaisten kehitysversioiden eli varianttien hallintaan. Mekaniikka-alalla taas versioita hallitaan peräkkäisinä versioina eli revisioina [47,48]. Mekaniikkatuotteiden informaationhallinta

perustuu, tiedostojen sijaan, oliopohjaiseen nimikerakenteeseen ja yksittäisten informaation palasten välisiin yhteyksiin [22,48].

Bergsjön (et al.) [47] mukaan edistyneet PDM-työkalut tarjoavat myös varianttien hallinnan toiminnallisuuden. Toiminnallisuus mahdollistaa tietyn ohjelmistoversion sisältävien tuotteiden jäljittämisen. Varianttien seuranta perustuu nimikkeiden välisiin yhteyksiin ja niiden historiatietoihin.

Ohjelmistojen versionhallinta PDM-työkalulla pitää siis toteuttaa oliopohjaista nimikerakennetta hyödyntäen. Ratkaisuksi Bergsjö (et al.) ehdottaa ohjelmiston jakamista luokkiin ja metodeihin, joita voidaan hallita hierarkkisesti olioina. PLC-ohjelmointikieliä käsittelevä standardi, IEC 61131-3 [48], sisältää olio-ohjelmointiin käytettävät ohjelmistokomponentit, kuten luokat ja metodit. Monet teollisuudessa käytetyt PLC-ohjelmointityökalut (esim. Siemensin Step 7) eivät kuitenkaan noudata standardia olio-ohjelmoinnin osalta [19]. Bergsjön (et al.) esittelemä lähestymistapa ei siis sellaisenaan sovellu tähän tutkimukseen.

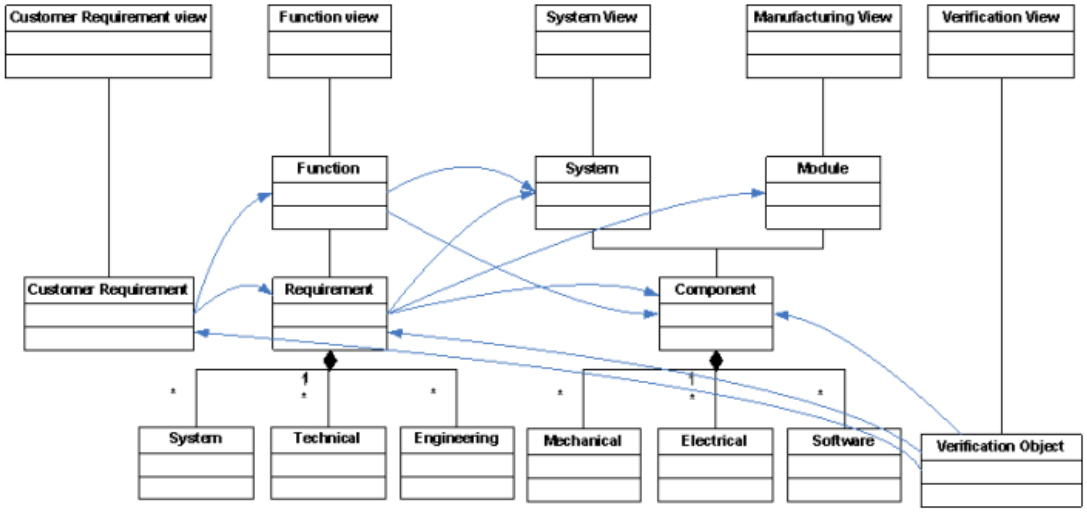
Myös Pernelle (et al.) [49,50] käsittelee artikkeleissaan mekatronisen systeemin PDM-integraatiota. Näissä artikkeleissa ehdotetaan PLC-ohjelmiston jaottelua funktiolohkoihin ja muuttujiin. Lähestymistapa voisi olla käyttökelpoinen modulaarisille PLC-ohjelmistoille. Pernellen (et al.) esittelemä tuoterakenne koostuu kolmesta eri nimikerakenteesta, jotka ovat mekaaninen nimikerakenne, sähkötekninen nimikerakenne ja ohjelmiston nimikerakenne. Kuhunkin nimikerakenteeseen liitetään alakohtainen dokumentaatio, kuten CAD-dokumentti, piirikaavio tai ohjelmistomoduulin koodi. Pernelle (et al.) erittelee nimikkeiden väliset yhteydet rakenteellisiin linkkeihin ja riippuvuuslinkkeihin. Rakenteelliset linkit ovat saman nimikerakenteen nimikkeiden välisiä yhteyksiä. Riippuvuuslinkit ovat eri nimikerakenteiden nimikkeiden välisiä yhteyksiä. Tuoterakenne on esitetty kuvassa 19.




***Kuva 19.*** *Mekatronisen systeemin tuoterakenne [50].*

Pernellen (et al.) tutkimuksissa ei kuitenkaan kerrota, miten edellä esitetyt linkit tulisi valita. Talkhestani [51], sen sijaan, esittelee tutkimuksessaan mekatronisen tuotteen riippuvuuslinkkejä. Nämä riippuvuuslinkit ovat alakohtaisten (mekaniikka-, sähkö- ja ohjelmistosuunnittelu) mallien välisiä rajapintoja. Rajapinnat voidaan määrittää niin kutsuttujen ankkuripisteiden mukaan. Ankkuripisteet kuvaavat mallien ominaisuuksia, joilla on riippuvuuksia toisten mallien ominaisuuksiin. Ankkuripisteiden välille muodostuu siis riippuvuuslinkki eli rajapinta. Esimerkkejä ankkuripisteistä ovat CAD-mallien geometriatiedot, piirikaavioiden kaapelointitiedot sekä funktiolohkojen I/O-muuttujat. Ankkuripisteiden määrittely helpottaa mekatronisen tuotteen muutostenhallintaa ja uudelleenkonfigurointia.

Bergsjön (et al.) mukaan mekatronisen systeemin tuotekehitystä voidaan tukea sopivien näkymien (engl. view) avulla. Artikkelissa esitellään viisi näkymää, jotka ovat asiakasvaatimusnäkymä, toimintonäkymä, systeeminäkymä, tuotantonäkymä ja verifiointinäkymä. Näkymien rakennekaaviota on havainnollistettu kuvassa 20. Kuvassa on esitetty myös näkymien väliset jäljitettävyyslinkit nuolilla.




***Kuva 20.*** *Näkymien rakennekaavio [47].*

Näkymät mahdollistavat dokumenttien liittämisen tuotekehitysprosessin eri vaiheisiin, joten PDM-työkalulla voidaan tukea mallipohjaista suunnittelua. Toiminnalliset vaatimukset (toimintonäkymä) johdetaan asiakasvaatimusten pohjalta (asiakasvaatimusnäkymä). Testausvaiheessa, näiden vaatimusten toteutuminen todennetaan simulointitulosten ja laitoshyväksyntätestien perusteella (verifiointinäkymä). Tämä vastaa luvussa 3.1.2 esitettyä V-mallia.

Systeeminäkymä ja tuotantonäkymä eroavat toisistaan tuoterakenteen esitystavan osalta. Systeeminäkymä vastaa luvussa 3.2.2 esitettyä hierarkkista systeemimallia. Systeeminäkymän tuoterakenne kuvaa siis moduulien välisiä toiminnallisia vuorovaikutuksia. Systeeminäkymässä ohjelmisto esitetään oliopohjaisesti, yksittäisiä toimintoja vastaavien, funktiolohkojen ja rajapintasignaalien avulla. Systeeminäkymä siis tukee simulointimallin konfigurointia ja ohjelmistojen modularisointia.

Tuotantonäkymän rakenne perustuu moduulien välisiin fyysisiin yhteyksiin. Tuotantonäkymä vastaa siis systeemin (fyysistä) kokoonpanoa. Näkymässä ohjelmistoprojekti kohdistetaan siihen liittyvään logiikkaohjainyksikköön. Näin ollen tuotantonäkymä osoittaa, mihin PLC-yksikköön ohjelmisto ladataan käyttöönottovaiheessa.

# 4. TAPAUSTUTKIMUS

## 4.1 Tapauksen esittely

Tämän työn keskeinen sisältö on tapaustutkimus, johon kirjallisuuskatsauksen ratkaisumalleja sovelletaan. Tutkittavana systeeminä on kiinteän kierrätyspolttoaineen (engl. Solid Recovered Fuel, SRF) valmistukseen käytettävä tuotantolinja. SRF on polttoainetta, jota valmistetaan murskaamalla ja kuivaamalla kiinteää jätettä. Tämän luvun tarkoituksena on luoda yleiskuva tapaustutkimuksen kontekstista. Aluksi esitellään tutkimuksen toimeksiantajana toimiva yritys ja sen toimiala. Tämän jälkeen, tutustutaan tapaustutkimuksen toteutukseen sekä tutkimuksen kohteena olevaan SRF-tuotantolinjaan. Lopuksi kuvaillaan tapaustutkimuksen lähtötilannetta ja tutkimuksen keskeisimpiä tavoitteita.

### 4.1.1 Taustatietoa

Tämän tutkimuksen tilaajayritys on BMH Technology, josta käytetään jatkossa nimeä BMH. Yritys toimittaa kestävää kehitystä tukevia ratkaisuja polttoaineen valmistukseen ja käsittelyyn. Biomassa- ja jätepohjaisilla polttoaineilla korvataan fossiilisia polttoaineita (kuten kivihiiltä) voimalaitoksissa sekä sementtiteollisuudessa. BMH on toimittanut maailmanlaajuisesti yli 200 polttoaineen tuotantosysteemiä. Näiden systeemien arvioidaan vähentäneen hiilidioksidipäästöjä yhteensä noin 441 miljoonalla tonnilla, vuodesta 1980 lähtien. Yritys tarjoaa asiakkailleen kokonaisvaltaisia ratkaisuja avaimet käteen –periaatteella. Toimitukset kattavat kaiken suunnittelusta rakentamiseen ja käyttöönottoon. Lisäksi käyttöönotettuja laitoksia tuetaan huolto- sekä palveluliiketoiminnalla.

Tapaustutkimuksessa keskitytään SRF-tuotantolinjaan. SRF:llä on korkea energian saantoarvo, joka on lähellä kivihiilen lämpöarvoa. SRF-polttoainetta voidaan siis käyttää kustannustehokkaasti lämmön ja sähkön tuotantoon. SRF-polttoaineen valmistukseen käytettäviä raaka-aineita ovat esimerkiksi kiinteä yhdyskuntajäte ja teollisuusjäte, tekstiilit ja muovit sekä käytetyt autonrenkaat. SRF on myös ratkaisu jätteiden hävittämiseen, sillä kuivattu ja vaaraton jäte muutetaan energiaksi polttamalla.

Heterogeenisen jätepohjaisen raaka-aineen jalostaminen polttokelpoiseksi aineeksi vaatii useita prosessivaiheita. Prosessin päävaiheet ovat raaka-aineen murskaus ja kierrätettävien materiaalien erottelu materiaalivirrasta. Erottelutekniikat perustuvat magneetteihin, hienoseuloihin, pyörrevirtoihin tai ilmaluokitteluun. Näin voidaan erotella esimerkiksi arvokkaita metalleja, lasia tai kovia muoveja pois poltettavasta jätteestä. Erotellut materiaalit toimitetaan uudelleenkäytettäväksi.

Edistyneillä laitteilla ja automatisoinnilla voidaan parantaa polttoaineen tuotannon turvallisuutta ja taloudellisuutta sekä laatua. Tässä tutkimuksessa käsitelläänkin tällaisia automaatiotoimintoja toteuttavia ohjelmistoja. Erityisesti käsitellään SRF-tuotantolinjoihin liittyvien automaatio-ohjelmistojen kehittämistä ja versionhallintaa.

Tutkimusaiheen valinnan taustalla on yrityksen visio uusien liiketoimintamahdollisuuksia luomisesta digitaalisten tuotteiden ja palveluiden tarjonnalla. Ohjelmistoilla voidaan tehdä liiketoimintaa myymällä ohjelmistoratkaisuja ja -päivityksiä asiakkaille. Systemaattiset ohjelmistokehitys- ja versionhallintamenettelyt ovat edellytyksiä ohjelmistoliiketoiminnalle.

Tämän tutkimuksen tarkoituksena on selvittää, miten tuotantolinjan ja ohjelmistojen modulaarisuus sekä versionhallintamenettelyt tukevat yrityksen liiketoimintaa. Tutkittavan yrityksen tapauksessa, modulaarisuuskonseptin soveltamisen haasteita ovat asiakaskohtaisen räätälöinnin tarve sekä projektiluontoiset suuren kokoluokan toimitukset. Myöskään versionhallinnan ratkaisut eivät ole yleispäteviä, vaan parhaiten soveltuvat käytännöt vaihtelevat yrityskohtaisesti.

### 4.1.2 Tutkimuksen toteutus

Tämä tutkimus on toteutettu tapaustutkimuksena yksittäisestä SRF-tuotantolinjasta. Tapaustutkimuksen näkökulmana on sekä poikittaissuuntainen että pitkittäissuuntainen läpileikkaus tuotantolinjasta. Pitkittäissuuntaisen läpileikkauksen tarkoituksena on kuvata tuotantolinja kokonaisuudessaan karkealla tasolla. Poikittaissuuntaisella läpileikkauksella tarkoitetaan tilannetta, jossa tuotantolinjasta on valittu pienenempi osakokonaisuus yksityiskohtaisempaan tarkasteluun. Läpileikkausstrategian avulla, tutkimuskohteeseen voidaan soveltaa luvussa 3.2.2 kuvattua hierarkkista mallinnusta.

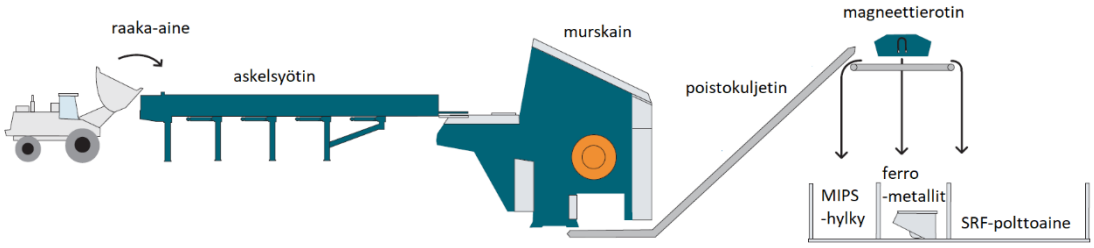
Tapaustutkimuksessa käytettyjä tutkimusmenetelmiä ovat tapaukseen liittyvän dokumentaation ja kirjallisuuden analysointi sekä aktiivinen osallistuva havainnointi. Havainnointi suoritettiin osallistumalla tuotekehitystyöhön aktiivisena toimijana yhteistyössä tuotekehitysorganisaation kanssa. Järjestely mahdollisti suoran yhteydenpidon yrityksen henkilöstöön sekä pääsyn tapaustutkimukseen liittyvään, yrityksen sisäiseen, materiaaliin. Tutkijan roolina oli kuitenkin organisaation ulkopuolinen toimija. Näin säilytettiin objektiivinen näkökulma tutkimuskohteeseen ja tutkimuskohteen kontekstina olevaan organisaatioon.

Tapaustutkimuksessa käytetty aineisto perustuu pääosin yrityksen tuottamiin ja toimittamiin teknisiin dokumentteihin. Tällaisia dokumentteja ovat esimerkiksi CAD-piirustukset, piirikaaviot, toiminnankuvaukset sekä käyttö- ja huolto-ohjeet. Dokumenttien sisältämät

tiedot ovat luottamuksellisia. Tapaukseen liittyviä tietoja ja tuloksia on tässä työssä anonymisoitu. Anonymisoinnin tarkoituksena on suojata yrityksen immateriaalioikeuksia. Yrityksen sisäisen materiaalin lisäksi kirjallisuuskatsauksessa esiteltyjä ratkaisumalleja on sovellettu tapaustutkimuksen tuotantolinjaan. Alan kirjallisuuden ja standardien ohella tapauksen aineistoa on täydennetty myös muilla julkisilla dokumenteilla, kuten ei-tieteellisillä artikkeleilla. Myös vapaamuotoiset keskustelut yrityksen edustajien kanssa ovat olleet tärkeä tiedonlähde tutkimuksessa.

### 4.1.3 SRF-tuotantolinja

Tässä luvussa tutustutaan tutkittavan tuotantolinjan toimintaperiaatteeseen. Luvussa käsitellään jätteenkäsittelyprosessin vaiheet pääpiirteittäin ja esitellään yksinkertaistettu kuvaus materiaalin etenemisestä tuotantolinjalla. Tuotantoprosessi on esitetty kuvassa 21.

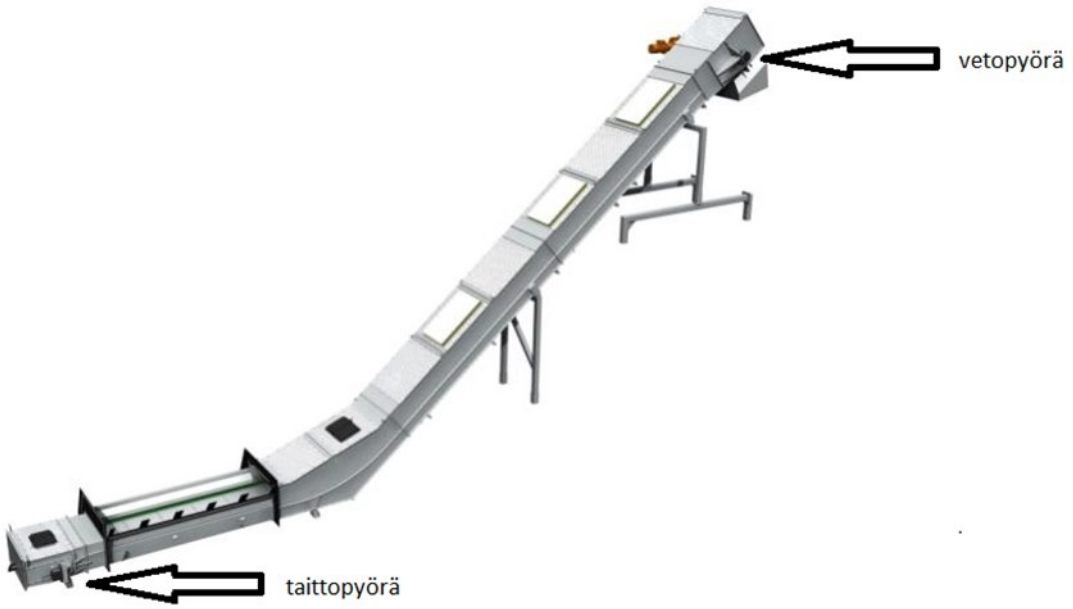



***Kuva 21.*** *Yksinkertaistettu prosessikaavio.*

Tuotantoprosessin ensimmäinen vaihe on raaka-aineen lastaus askelsyöttimeen. Askelsyöttimen tehtävänä on säätää murskaimelle syötettävän materiaalin määrää. Näin tuotantoprosessin kapasiteetti voidaan pitää halutulla tasolla. Askelsyötin toimii siis puskurina, joka varmistaa jatkuvatoimisen syöttövirtauksen ja raaka-aineen riittävyyden. Puskuri pidentää täyttöväliä, vapauttaen operaattorin muihin työtehtäviin täyttöjen välillä.
Murskain voidaan ajatella tuotantoprosessin pullonkaulana, jonka kapasiteetti rajoittaa tuotantolinjan materiaalivirtausta. Murskaimen kuormituksella voidaan näin ollen vaikuttaa tuotannon määrään ja ajoitukseen. Toisin sanoen, murskain toimii tuotannon ohjauspisteenä.
Tuotantoprosessin toinen vaihe on siis raaka-aineen murskaus. Raaka-aine hienonnetaan tasaisen pieniksi partikkeleiksi hydraulisesti säädettävillä leikkuuterillä. Pienempi partikkelikoko parantaa materiaalien erottelutarkkuutta ja edistää polttoaineen palamista. Murskautumaton materiaali poistetaan murskaimesta automaattisesti, niin kutsutulla MIPS-ominaisuudella (engl. Massive Impulse Protection System, MIPS). Ominaisuudella suojataan murskainta kiinteiltä metallikappaleilta, jotka voivat aiheuttaa vahinkoa laitteelle. MIPS-ominaisuus tunnistaa kappaleen, joka ei murskaudu terien välissä. Tällöin

materiaalin syöttö keskeytetään ja murskaimessa oleva materiaali pudotetaan poistokuljettimelle. Pienen viiveen jälkeen, materiaalin syöttö murskaimelle jatkuu automaattisesti. Materiaalin poiston automatisointi parantaa prosessin turvallisuutta, sillä operaattorin väliintuloa ei tarvita.

Sekä murskattu että murskautumaton materiaali putoavat murskaimelta poistokuljettimelle. Poistokuljettimen tehtävänä on kuljettaa materiaalia murskaimelta magneettierottimelle. Poistokuljetin on suljettu ja pölytiivis ketjukuljetin. Sen kalteva kuljetinketju mahdollistaa materiaalin kuljettamisen jyrkässä nousukulmassa. Poistokuljetin koostuu kuljetinketjun lisäksi veto- ja taittopyöristä sekä runkorakenteesta. Vetopyörä saa aikaan kuljetinketjun liikkeen. Taittopyörällä vaihdetaan ketjun kulkusuuntaa. Poistokuljettimen rakenne on esitetty kuvassa 22.




***Kuva 22.*** *Tutkittavan poistokuljettimen rakenne, julkaistu BMH:n luvalla.*

Magneettierotin koostuu hihnakuljettimesta ja magneetista. Vahvan magneetin avulla, materiaalivirrasta erotellaan magneettiset metallit. Tällöin ferromagneettiset metallit (kuten rauta) tarttuvat kiinni magneettihihnaan. Ei-magneettiset partikkelit kulkevat reagoimatta magneettierottimen läpi. Hihnakuljettimen hihna voi liikkua sekä eteen- että taaksepäin. Kääntyvän hihnan avulla, murskautumaton materiaali poistetaan prosessista ennen magneettisten metallien erottelua. Suuret metallipartikkelit eivät siis jää jumiin hihnakuljettimen ja magneetin väliin.

Erottelun jälkeen, polttoaineeksi kelpaava materiaali kuljetetaan polttoainevarastoihin. Varastot toimivat puskurina SRF-polttoaineen polttoprosessille. Puskurin avulla tasataan polttoainevirtaa ja vastataan vaihtelevaan polttoaineen tarpeeseen.

### 4.1.4 Ohjelmistokehityksen nykytilanne yrityksessä

Yrityksellä ei ole tällä hetkellä käytössä systemaattista versionhallintaa automaatio-ohjelmistoille. Ohjelmistojen tallennusvälineet ja säilytyspaikat vaihtelevat. Kaikista ohjelmistoista ei myöskään löydy viimeisintä, käyttöönotetun version, kopiota.
Automaatiotoimintoja tai ohjelmistojen implementointeja ei ole kuvattu riittävällä tarkkuudella. Ohjelmiston toiminnan määrittely jää tällöin liiaksi implementointitiimin vastuulle. Ohjelmoijilla on siis liikaa vapauksia, jolloin projektikohtaiset implementoinnit eroavat toisistaan.
Myös suunnittelu- ja käyttöönottotietojen haltuunotossa on parantamisen varaa. Ohjausjärjestelmien I/O-listat ovat puutteellisia, eikä käyttöönotettavia parametreja ole dokumentoitu. Nämä puutteet vaikeuttavat ohjelmistojen testaamista ja käyttöönottoa.
Ohjelmistojen valmisasteen seurantaan ja ohjelmistokehityksen aikataulutuksen laadintaan tulisi panostaa enemmän. Tällä hetkellä ohjelmointityö jää liiaksi käyttöönottovaiheeseen, mikä lisää kehityskustannuksia ja aiheuttaa suuria aikataulupaineita.
Tämän tutkimuksen tavoitteena onkin tuoda esiin uusia käytäntöjä ja parannusehdotuksia, jotka laskevat ohjelmistokehitykseen liittyviä kustannuksia. Erityisesti tutkitaan, kuinka mallipohjaista suunnittelua ja simulointia voidaan käyttää ohjelmistojen kehittämisessä. Mallipohjainen suunnittelu vaatii tuotantolinjan ja siihen liittyvän ohjelmiston modularisointia. Mallien ja ohjelmistojen uudelleenkäytettävyys edellyttää laadukkaan systeemisuunnittelun lisäksi modulaariselle systeemille soveltuvaa versionhallintastrategiaa. Modulaarisuus ja versionhallinta ovatkin tapaustutkimuksen pääteemoja.
Tämä tapaustutkimus voidaan jakaa pääpiirteittäin kolmeen osaan. Ensimmäisessä osassa tutkittavalle tuotantolinjalle luodaan modulaarinen tuotetietorakenne. Tämän jälkeen, tutkitaan modulaarisen tuotantolinjan versionhallintastrategiaa. Lopuksi laaditaan toimenpide-ehdotus ohjelmistokehityksen edistämiseksi, tapaustutkimuksen havaintojen pohjalta. Toimenpide-ehdotus sisältää ohjelmistokehitykseen ja versionhallintaan soveltuvien työkalujen määrittelyn sekä liiketoimintavaikutuksien arvioinnin ja investointiehdotuksia.

## 4.2 Tuotantolinjan modularisointi

Tässä luvussa esitetään tuotantolinjan modulaarinen systeemikuvaus. Hierarkkinen kuvaus etenee korkealta abstraktiotasolta kohti yksityiskohtaista, matalan abstraktiotason, kuvausta. Lähestymistapa mahdollistaa kompleksisen tuotantolinjan jakamisen yksinkertaisempiin yksikköprosesseihin. Modulaarinen tuotetiedon kuvailu aloitetaan systeemitasolta, josta siirrytään laitteistotasolle ja lopulta ohjelmistotasolle.

### 4.2.1 Tuotantolinjan abstraktiotasojen kuvaus

Tässä luvussa esitetään hierarkkinen systeemikuvaus tuotantolinjalle. Pitkittäissuuntaisen läpileikkauksen mukaisesti, kokonaissysteemi jaetaan pienempiin osasysteemeihin eli moduuleihin. Näiden osasysteemien väliset rajapinnat kuvataan materiaalin ja informaation vaihtona siten, että yhden osasysteemin muutokset eivät vaikuta muihin osasysteemeihin rajapintojen pysyessä ennallaan.

Valittu esitystapa on lohkokaavio, jossa lohkot kuvaavat yksikköprosesseja. Yksikköprosessit ovat teollisuudenalasta riippumattomia kokonaisprosessin osia. Esimerkkejä mekaanisista yksikköprosesseista ovat kiinteän materiaalin murskaus, kuljetus ja erottelu. Yksikköprosesseja voidaan tarkastella yleispätevillä menetelmillä, kuten black box –tarkastelulla. Yksikköprosessien rajat on valittu vastaamaan kuvassa 21 esitettyjen mekaanisten prosessilaitteiden määrittämiä rajoja.

Lohkokaavion nuolet kuvaavat yksikköprosessien välisiä materiaali- ja informaatiovirtoja. Massatasetarkastelulla voidaan arvioida materiaalivirtojen suhteita. Menetelmä perustuu aineen häviämättömyyden lakiin.

Massataseet eli materiaalitaseet toimivat myös pohjana energiataseiden laskentaan. Energiataseilla voidaan estimoida prosessin energiantarvetta tai lämpöhäviöitä. Tässä tutkimuksessa energiavirtojen mallinnusta ei kuitenkaan käsitellä.

Materiaalitaseen kertymä $\Delta \mathrm{m}$ on yksikköprosessiin sisäänmenevien $m_{in}$ ja ulostulevien $m_{out}$ massojen välinen erotus:

$$\Delta \mathrm{m} = \sum m_{in} - \sum m_{out} \ . \tag{1}$$

Materiaalitaseen kertymää voidaan siis pitää systeemin dynaamisena tilana. Kertymän ollessa positiivinen, yksikköprosessiin kumuloituu materiaalia. Vastaavasti, kertymän ollessa negatiivinen prosessista poistetaan materiaalia. Materiaalitase voidaan esittää myös massojen muutosnopeuksien avulla muodossa:

$$\Delta \dot{\mathrm{m}} = \sum \dot{m}_{in} - \sum \dot{m}_{out} \ , \tag{2}$$

jossa pistenotaatio tarkoittaa aikaderivaattaa. Materiaalitaseen aikaderivaattoja kutsutaan tässä tutkimuksessa materiaalivirroiksi.

Kuvan 21 mukaisesti materiaalivirran komponentteja ovat MIPS-hylky, ferromagneettiset metallit ja SRF-polttoaine. Tutkittavan tuotantolinjan jätejakaumasta tai erottelun materiaalitaseesta ei ole saatavissa tarkkaa tietoa. Käytettävän raaka-aineen laatu ja koostumus vaihtelee huomattavasti maantieteellisestä sijainnista ja tarkasteluajankohdasta

riippuen. Tutkittava SRF-tuotantolinja on asennettu ruotsissa sijaitsevaan jätteenkäsittelylaitokseen. Näin ollen tarkasteltavaksi raaka-aineeksi on valittu keskimääräinen ruotsalainen kotitalousjäte. Ruotsalaiseen kotitalousjätteeseen liittyvää tilastotietoa esiteellään lähteessä [52]. Muut materiaalitaseen laskemiseen tarvittavat lähtötiedot ja oletukset perustuvat lähteiseen [53]. Taulukossa 1 on esitetty materiaalivirran massaosuudet raaka-ainekomponenteittain.

***Taulukko 1.*** *Raaka-aineen ominaisuudet.*

| **Raaka-aine** | **Massa-osuus (%)** | **Lähde** |
|---|---|---|
| romumetalli (MIPS-hylky) | 10 | [52] |
| ferromagneettiset metallit (FE) | 5 | [53] |
| kelpo polttoaine (SRF) | 85 | |
| yhteensä | 100 | |

Materiaalivirran massatase on mitoitettu tuotantolinjan pullonkaulan eli murskaimen käsittelykapasiteetin mukaan. Murskain pystyy käsittelemään kotitalousjätettä keskimäärin 35 tonnia tunnissa eli 9.72 kilogrammaa sekunnissa. Massataseen laskentaesimerkki on esitetty liitteessä A.

Informaatiovirta voidaan jakaa ohjauksiin ja tilatietoihin. Ohjaukset ovat aktivoivia informaatiokomponentteja, jotka käskevät toimilaitteita. Tilatiedot ovat passiivisia informaatiokomponentteja, jotka ilmaisevat prosessin tiloja. Turva-automaatiotoimintoihin, kuten lukituksiin, liittyvät ohjaukset ja tilatiedot ovat tavanomaisesti boolean-tyyppisiä muuttujia. Lukituksilla estetään vaaraa aiheuttavat toimenpiteet, kuten moottorin käynnistäminen vikatilanteessa. Lukitukset kuvaavat siis prosessilaitteiden toimintojen välisiä riippuvuuksia. Boolean on totuusarvoa kuvaava tietotyyppi, jolla on vain kaksi mahdollista tilaa (1 eli tosi tai 0 eli epätosi).

Tuotantolinjan informaatiokomponentit on valittu siten, että ne kuvaavat tuotantolinjan liikkeenohjausta lukitussekvenssien mukaisesti. Prosessilaitteita voidaan ohjata käynnistymään tai pysähtymään sekä liikkumaan eteen- tai taaksepäin. Tilatiedoilla määritellään, voidaanko tietty ohjaustoiminto toteuttaa. Informaatiokomponentit ja niiden tilat on esitetty taulukossa 2.

***Taulukko 2.*** *Informaatiovirran ohjaukset ja tilatiedot.*

| Muuttujan nimi | Tieto-tyyppi | Kuvaus |
|---|---|---|
| käynnistä | Bool | ohjaus asettaa moottorin päälle / pois päältä (1 = päälle, 0 = pois) |
| suuntaa | Bool | ohjaus asettaa liikkeen suunnan (1 = eteenpäin, 0 = taaksepäin) |
| käyntitieto | Bool | moottorin tilatieto (1 = käynnistetty, 0 = ei-käynnistetty) |
| suuntatieto | Bool | tilatieto liikkeen suunnasta (1 = eteenpäin, 0 taaksepäin) |
| vikaantumis-tieto | Bool | tilatieto toimintahäiriöistä (1 = ei vikoja, 0 = vähintään yksi vika) |

Kuvassa 23 on esitetty korkean abstraktiotason malli tuotantolinjasta. Tuotantoprosessi on kuvattu yhtenä black box –mallina, joka käyttää taulukoissa 1 ja 2 määriteltyjä rajapintoja. Materiaalirajapinnan maksimiarvo määräytyy tuotantolinjan käsittelykapasiteetin mukaan. Materiaalirajapinnan minimiarvo on nolla tarkoittaen, että materiaalia ei virtaa sisään systeemiin tai sieltä ulos. Materiaalivirran yksiköksi on, SI-järjestelmän (ransk. Système international d'unités) mukaisesti, valittu kilogrammaa sekunnissa (kg/s). Boolean-tyyppiset muuttujat saavat arvon nolla tai yksi. Tilatietojen muuttujat ovat dimensiottomia.

Informaatiovirta ja materiaalivirta voidaan esittää yhtälöillä:

$$informaatiovirta = \begin{bmatrix} käynnistä, suuntaa \\ käyntitieto, suuntatieto, vikaantumistieto \end{bmatrix} \quad (3)$$

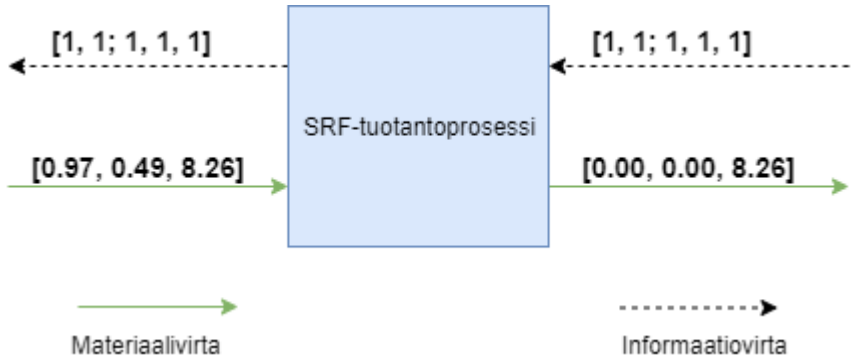
ja

$$materiaalivirta = [MIPShylky, FEmetallit, SRFpolttoaine] \; . \quad (4)$$

Informaatiovirta on siis ositettu matriisi, joka koostuu kahdesta vaakavektorista. ylempi vaakavektori sisältää ohjaukset ja alempi vaakavektori tilatiedot. Kuvassa 23, nämä vektorit on erotettu puolipisteellä.

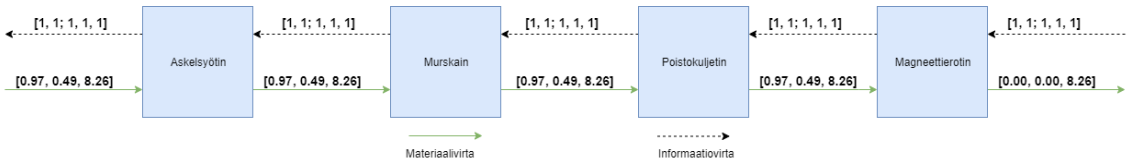
Materiaalivirran suunta on, kuvan 21 mukaisesti, vasemmalta oikealle. Kuvan 23 malli kuvaa siis käsittelyprosessia, jossa raaka-aineesta (kotitalousjäte) jalostetaan lopputuote (SRF-polttoaine). Informaatiovirta sen sijaan kulkee vastakkaiseen suuntaan. Informaatiovirta kuvaa siis tuotannonohjauksen periaatetta, joka pohjautuu imuohjaukseen. SRF-tuotantolinjan imuohjauksessa materiaalivirtaa ohjataan polttoaineen tarpeen

mukaan. Imuohjauksen tavoitteena on raaka-aine- ja polttoainevarastojen aiheuttamien kustannusten minimointi.




***Kuva 23.*** *Kokonaisprosessi ja sen rajapinnat.*

Kuvassa 24 on esitetty malli, jossa kokonaisprosessi on jaettu pelkistettyihin yksikköprosesseihin. Yksikköprosessien ominaisuudet on periytetty hierarkkisesti kuvan 1 kokonaisprosessista. Näin ollen myös yksikköprosessit ovat black box –malleja, jotka käyttävät samoja rajapintoja, yhtälöt (3) ja (4), kuin kokonaisprosessin malli.




***Kuva 24.*** *Tuotantolinjan jakaminen yksikköprosesseihin.*

Tuotantolinjan dynaamista käyttäytymistä voidaan kuvata materiaalivirtojen ja massakertymien avulla. Yhtälöiden (1) ja (2) perusteella voidaan päätellä, että tietyn yksikköprosessin dynaaminen tila on sisään- ja ulostulevien materiaalivirtojen välisen erotuksen integraali. Toisin sanoen materiaalirajapinta määrittelee yksikköprosessin tilamuuttujan aikaderivaatan, joten ulostulosignaali saadaan integroimalla tilamuuttujan poikkeamaa tasapainotilasta. Simulaattorissa tämä operaatio voidaan toteuttaa numeerisella integraattorilla.

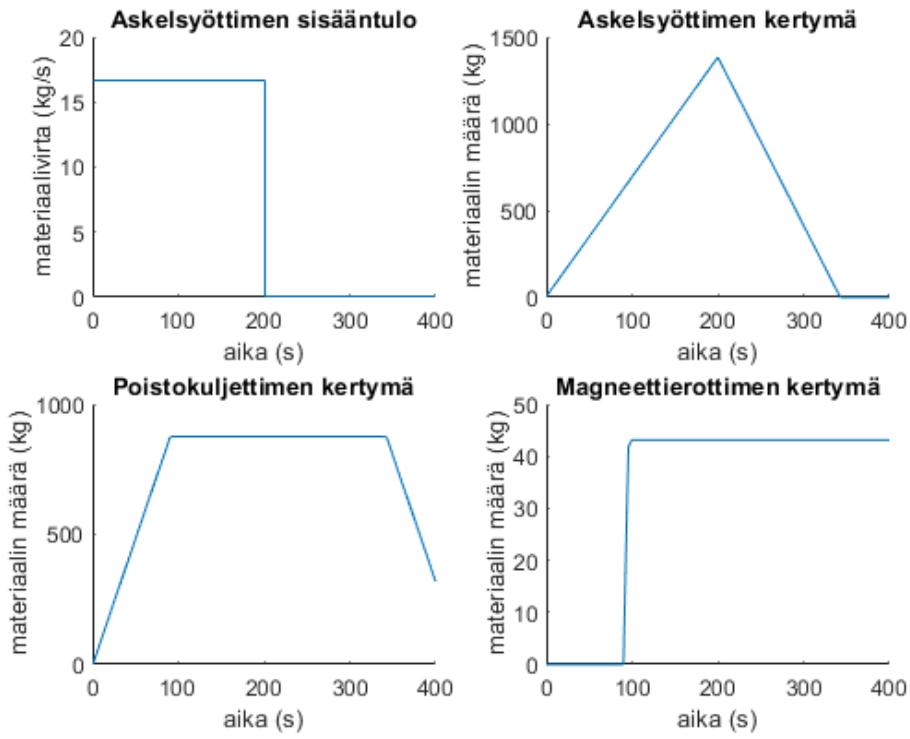
Black box –malleja voidaan käyttää simulaattorin kehityksessä. Kuvan 24 mukaisen abstraktiotason tarkoituksena on määritellä yleispätevä tietorakenne SRF-tuotantolinjan simulointiin. Vakioiduista rajapinnoista koostuva tuotantolinja on modulaarinen. Näin ollen yksikköprosessien järjestystä tai sisäisiä toteutuksia voidaan muuttaa ilman, että muutokset vaikuttavat muiden yksikköprosessien toteutukseen. Tietyn yksikköprosessin muutokset vaikuttavat muihin yksikköprosesseihin ainoastaan silloin, kun osaprosessien väliset rajapinnat muuttuvat.

Kuten edellä todettiin, materiaalivirta ei kuvaa tutkittavan systeemin todellista erottelutasetta. Liitteen A esimerkkilaskelmaa voidaan kuitenkin käyttää apuna simulaatiokokeen luomisessa. Esimerkkilaskelman arvoja voidaan tällöin verrata simulointikokeen tuottamiin materiaalivirran arvoihin. Näin voidaan varmistaa, että simulointitulokset ovat käyttökelpoisia.

Tuotantolinjan kuljettimille on määritetty vakionopeudet sekä eteenpäin että taaksepäin suuntautuvalle liikkeelle. Kuljettimia siis ohjataan valitsemalla haluttu vakionopeus ja suunta. Kuvassa 25 on esitetty simulointituloksia transienttikokeesta, jossa kuljettimia ajetaan eteenpäin niiden maksiminopeuksilla. Tuotantolinja on alkutilanteessa käynnissä tyhjänä. Operaattori täyttää askelsyötintä täydellä kapasiteetilla 200 sekunnin ajan, jonka jälkeen materiaalin tuonti syöttimelle keskeytetään. Askelsyötin syöttää murskainta automaattisesti siten, että murskain toimii maksimikapasiteetillaan. Prosessilaitteiden käsittelykapasiteetit sekä suurimmat sallitut kuormat ja nopeudet on esitetty taulukossa 3.

***Taulukko 3.*** *Prosessilaitteiden ominaisuudet.*

| **Prosessilaite** | **Käsittely-kapasiteetti (kg/s)** | **Maksimi-kuorma (kg)** | **Maksimi-nopeus (m/s)** |
|---|---:|---:|---:|
| askelsyötin | 16.67 | 15600.00 | - |
| murskain | 9.72 | - | - |
| poistokuljetin | 28.89 | 2600.00 | 0.25 |
| magneettierotin | 28.89 | 115.56 | 1.00 |

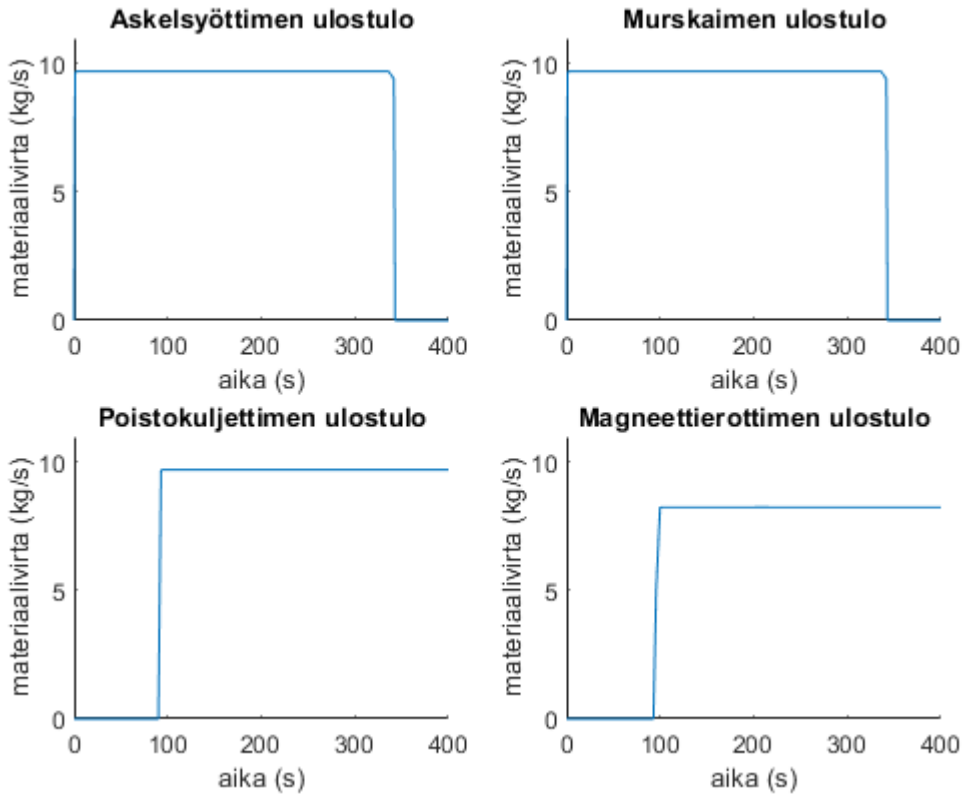



***Kuva 25.*** *Massakertymät vakionopeusajolla.*

Kuvasta 25 nähdään, että materiaalin kulkuaika poistokuljettimen läpi on noin 90 sekuntia. Poistokuljettimelta materiaali siirtyy magneettierottimelle, jossa materiaalin läpimenoaika on 4 sekuntia. Nämä viiveet aiheuttavat materiaalien kumuloitumisen kuljettimiin. Kuvasta 25 huomataan myös, että materiaalin loppuessa poistokuljettimen massakertymä alkaa pienentyä. Muutos ei kuitenkaan näy magneettierottimen kertymässä, tarkasteltavalla aikavälillä. Syötettävän materiaalin loppuminen näkyisi magneettierottimen sisääntulossa vasta poistokuljettimen kulkuajan aiheuttaman viiveen jälkeen. Magneettierottimen kertymä lähtisi siis laskuun vasta 90 sekuntia myöhemmin, verrattuna poistokuljettimeen.

Edellä kuvatut materiaalin kuljetuksen aiheuttamat viiveet on helppo havaita, tarkastelemalla yksikköprosessien ulostuloja. Nämä ulostulot on esitelty kuvassa 26. Simulointimallissa murskain on oletettu ideaaliseksi, jolloin raaka-aineen murskauksesta aiheutuvaa viivettä ei ole mallinnettu. Myös MIPS-toimenpiteen aiheuttama viive on jätetty mallintamatta. MIPS-toimenpide pysäyttää materiaalin syötön murskaimelle hylkymateriaalin poistamisen ajaksi. Tätä aikaväliä voidaan kutsua nimellä MIPS-aika. Poistokuljetin kuljettaa hylkymateriaalin magneettierottimelle, jossa hylkymateriaali poistetaan prosessista. Hylkymateriaalin saapuessa hihnakuljettimelle, hihnan kulkusuunta vaihtuu. Tällöin hihna liikkuu taaksepäin, mitattua MIPS-aikaa vastaavan sekuntimäärän ajan. Tä-

män jälkeen, hihnakuljettimen kulkusuunta vaihtuu takaisin alkuperäiseen. Toisin sanoen, hihnakuljetin kulkee jälleen eteenpäin. Todellisuudessa, kuljettimien viiveeseen tulisi siis lisätä myös MIPS-aika ja murskausviive.




***Kuva 26.*** *Vasteet vakionopeusajolla.*

Vertaamalla kuvia 25 ja 26 taulukkoon 3 huomataan myös, että poistokuljettimen ja magneettierottimen kuljettamat materiaalimäärät ovat selkeästi niiden maksimikuormia ja käsittelykapasiteettia pienempiä. Materiaalin kuljettamiseen riittäisi siis maksiminopeuksia pienemmät nopeudet. Kuljettimien nopeuksia voitaisiin siis optimoida energiankulutuksen minimoimiseksi. Materiaalimäärien ja polttoaineentarpeen mukaan säädettävän nopeuden konsepti esitellään luvussa 4.2.2.

Seuraavassa alaluvussa määritellään kolmas ja neljäs abstraktiotaso. Nämä abstraktiotasot ovat alustariippuva malli ja osasysteemin yksityiskohtaisen kuvaus. Pitkittäissuuntaisesta läpileikkauksesta siirrytään siis poikittaissuuntaiseen läpileikkaukseen. Tarkasteltavaksi osasysteemiksi on valittu poistokuljetin. Poistokuljetin on yksinkertainen ja havainnollistava esimerkkitapaus tuotantolinjaan liittyvän tuotetiedon mallinnukseen.

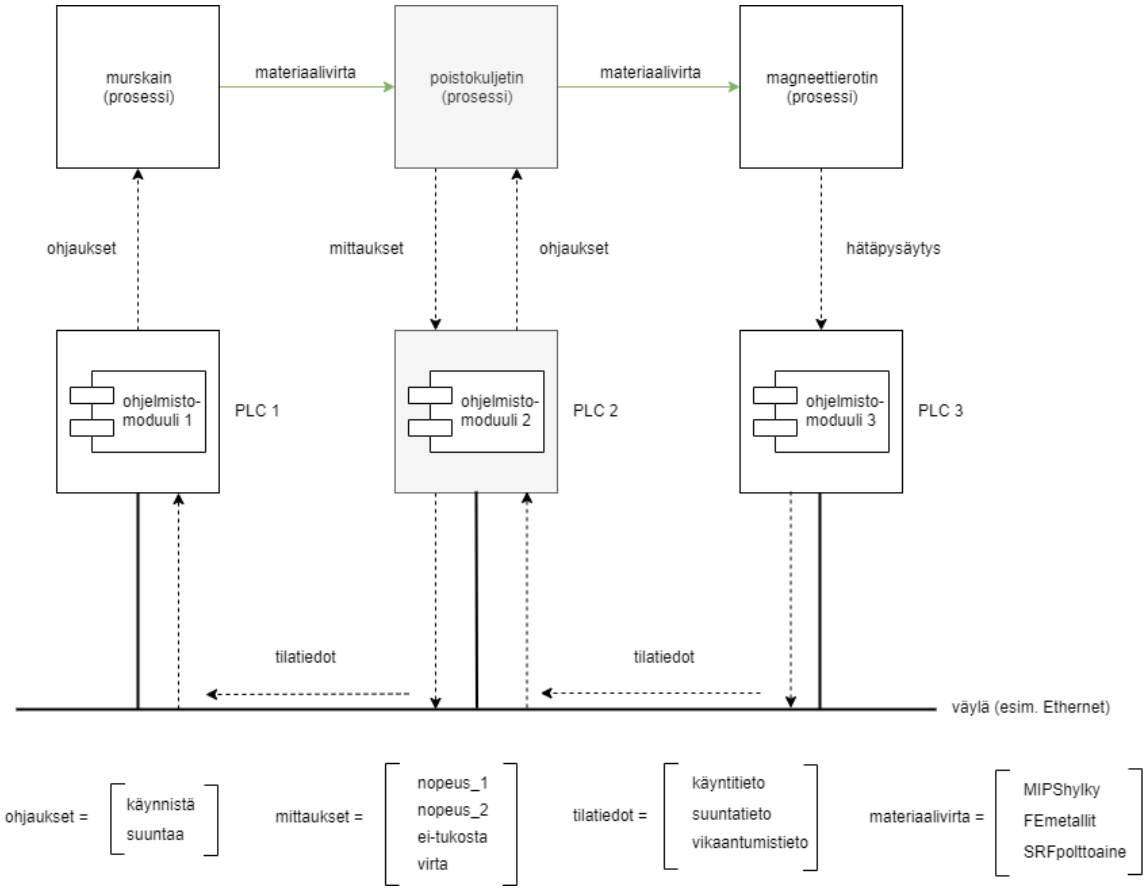
### 4.2.2 Poistokuljettimen yksityiskohtainen kuvaus

Yksityiskohtaista osasysteemin mallia voidaan hyödyntää simulointimallien ja ohjelmistomoduulien kehityksessä. Lisäksi tätä mallia voidaan edelleen käyttää PLC-ohjelmiston koodimallin sekä HIL-simuloinnin testimallin luomisessa. Tällöin toiminnalliseen malliin on lisättävä käyttäytymistä kuvaava looginen malli.

Tässä tutkimuksessa on kuitenkin mallinnettu vain toimintolohkot ja niiden rajapinnat kuvaava toiminnallinen malli. Prosessimoduulien tai ohjelmistomoduulien sisäiseen toteutukseen ei oteta kantaa, vaan ne on mallinnettu black boxeina. Myöskään rajapintojen toteutusta ei määritellä. Näin ollen mallinnuksen tarkoituksena on määritellä moduulien välisten rajapintojen informaatiosisältö siten, että tuotantolinjan rakenteen kuvaus tukee modulaarista ohjelmointia. Modulaarisessa ohjelmoinnissa, ohjelmiston toiminnot jaetaan itsenäisiin moduuleihin. Nämä ohjelmistomoduulit kommunikoivat keskenään hyvin määriteltyjen rajapintojen kautta. Moduulit vaativat ja tarjoavat rajapintoja, jotka ovat näkyvissä muille moduuleille. Moduulin sisäinen toteutus on kuitenkin piilotettu muilta moduuleilta. Modulaariseen ohjelmointiin voidaan käyttää olio-ohjelmoinnin mekanismeja.

Alustariippuva malli kuvaa laitteiston ja ohjelmiston sekä niiden väliset rajapinnat. Alustariippuvalla mallilla voidaan määritellä, miten ohjelmiston toiminnallisuudet liittyvät sähköautomaatiolaitteisiin ja mekaniikkalaitteisiin. Alustariippuva malli soveltuu hyvin, laitteistosta ja ohjelmistosta koostuvan, tuotantolinjan modulaarisen rakenteen kuvailutavaksi.

Kuvassa 27 on esitetty alustariippuva malli, joka kuvaa poistokuljettimen liikkeenohjausta. Alustariippuvan mallin ohjelmistomoduulit on valittu vastaamaan mekaanisiin yksikköprosesseihin jaetun tuotantolinjan modulaarista rakennetta. Ohjelmiston toiminnallisuudet on siis hajautettu erillisiksi säätöpiireiksi yksikköprosesseittain. Hajautus helpottaa systeemin ylläpitoa ja muutostenhallintaa. Toimintojen hajautus myös lisää systeemin luotettavuutta sekä parantaa skaalautuvuutta. Hajautetut PLC-laitteet kommunikoivat keskenään väyläsignaalien (tilatiedot) välityksellä. Prosessimoduulin ja ohjelmistomoduulin väliset prosessisignaalit (mittaukset ja ohjaukset) kulkevat tulo- ja lähtöpiirejä pitkin. Signaalien informaatiosisältöjen kuvaus on esitelty liitteessä B.




***Kuva 27.*** *Poistokuljettimen alustariippuva malli.*

Kuvan 27 PLC-laitteet voivat olla (esimerkiksi) prosessiaseman keskusyksiköitä (engl. Central Processing Unit) tai hajautettuja I/O-yksiköitä. Hajautetuilla I/O-yksiköillä, tulo- ja lähtöpiirit voidaan viedä prosessiaseman luota lähemmäs toimilaitteita. Toisaalta, PLC-laitteiden toiminnallisuudet voitaisiin toteuttaa myös pöytätietokoneella (engl. desktop computer). Tällöin PLC-laitteet virtualisoitaisiin pöytätietokoneeseen asennettavalla emulaattoriohjelmistolla. Tässä tutkimuksessa ei kuitenkaan oteta kantaa hajautuksen käytännön toteutustapaan.

Informaatiovirta koostuu ohjauksista, mittauksista ja tilatiedoista. Ohjaukset ja tilatiedot on periytetty korkeamman abstraktiotason mallilta. Mittaukset ovat tutkittavan yksikköprosessin, poistokuljettimen, antureiden lähettämiä mittaustietoja.

Kuvasta 27 nähdään, että ohjelmistomoduulit kommunikoivat keskenään paketeilla, jotka kokoavat yhteen useita informaatiokomponentteja. Rajapintamuuttujien kokoaminen yhteen, vektorimuotoisiksi muuttujiksi, helpottaa rajapintojen hallintaa ja ohjelmiston ylläpidettävyyttä. Oikein valittujen ja hyvin määriteltyjen rajapintojen avulla, ohjelmiston rakentamiseen voidaan tarjota uudelleenkäytettäviä moduuleja. Ratkaisu siis tukee modulaarista ohjelmointia.

Poistokuljettimen toimintoja ovat käynnistys ja pysäytys sekä eteenpäin ja taaksepäin ajo. Poistokuljettimen nopeutta ohjataan taajuusmuuttajakäytöllä. Poistokuljettimen nopeudet on määritelty taajuusmuuttajan ohjelmoitavissa parametreissa. Logiikkaohjaimella (PLC) voidaan ohjata taajuusmuuttajaa valitsemalla suunta ja nopeus. Taajuusmuuttaja ohjaa vaihtovirtamoottorin pyörimisnopeutta ja -suuntaa.

Poistokuljettimen suojaus- ja turvatoimintoihin käytettäviä laitteita ovat pyörintävahti ja tukosvahti. Pyörintävahdin tehtävänä on valvoa poistokuljettimen mekaanista kuntoa. Pyörintävahti koostuu induktiivisesta pulssianturista ja rajakytkimestä. Esimerkiksi kuljettimen ketjun katketessa, pulssianturi ei saavuta vaadittua minimipulssimäärää. Tällöin rajakytkin lähettää mittaustiedon logiikkaohjaimelle, joka pysäyttää poistokuljettimen. Myös poistokuljetinta edeltävät prosessilaitteet pysäytetään lukitusketjun mukaisesti.

Pyörintävahdilla siis seurataan, että kuljettimen moottori pyörii riittävän suurella nopeudella. Tämä nopeuden raja-arvo määritetään pyörintävahdin ohjelmoitavissa parametreissa. Poistokuljettimella on kaksi pyörintävahtia, jotka sijaitsevat taittopyörän akselin kummassakin päässä.

Tukosvahti on asennettu (vetopäähän) kuljettimien risteyskohtaan, jossa materiaali siirtyy poistokuljettimelta magneettierottimen hihnakuljettimelle. Poistokuljettimen tukosvahtina on kalvokytkin, jolla valvotaan materiaalin pinnankorkeutta. Tukos aiheuttaa materiaalin pinnankorkeuden nousemisen. Tällöin anturin tuntoelimenä toimiva kalvo taipuu, materiaalin aiheuttaman paineen vaikutuksesta. Kalvon taipuminen aktivoi mekaanisen rajakytkimen, mistä tulee mittaustieto logiikkaohjaimelle. Tällöin PLC pysäyttää poistokuljettimen sekä edeltävät prosessilaitteet lukitusketjun mukaisesti. Tukosvahdin aktivoitumisherkkyyteen voidaan vaikuttaa säätöruuvin asennolla.

Pyörintävahtien ja tukosvahdin mittaustietoja käytetään poistokuljettimen sisäisissä lukituksissa. Poistokuljettimen ulkoisena lukituksena on magneettierottimen vikaantumistieto. Vikaantumistietoja voidaan edelleen käyttää hälytysten generointiin. Hihnakuljetin on varustettu vaijerikytkimellä, jonka aktivoiminen aiheuttaa hätäpysäytyksen. Hätäpysäytystieto välitetään poistokuljettimen säätimelle tilatiedot-muuttujan avulla. Poistokuljetin pysäytetään ja tilatiedot välitetään edelleen eteenpäin murskaimen säätimelle.

Poistokuljettimessa on myös virtamittari, jolla seurataan moottorin mahdollista ylikuormitusta. Äkilliset virtapiikit voivat vahingoittaa kuljetinta tai aiheuttaa tulipalon. Virta mitataan suoraan taajuusmuuttajasta ja lähetetään analogiasignaalina logiikkaohjaimelle. Mitatun virta-arvon ylittäessä määritetyn raja-arvon, PLC pysäyttää moottorin ja lähettää vikaantumistiedon eteenpäin murskaimen säätimelle.

Logiikkaohjaimen laitteisto on modulaarinen. Laitteiston moduuleja ovat prosessorimoduuli, I/O-moduulit, ja virtalähdemoduuli. Prosessimoduuli sisältää keskusyksikön ja muistikortin. I/O-moduulit koostuvat I/O-korteista. I/O-kortit voidaan jaotella digitaalisiin

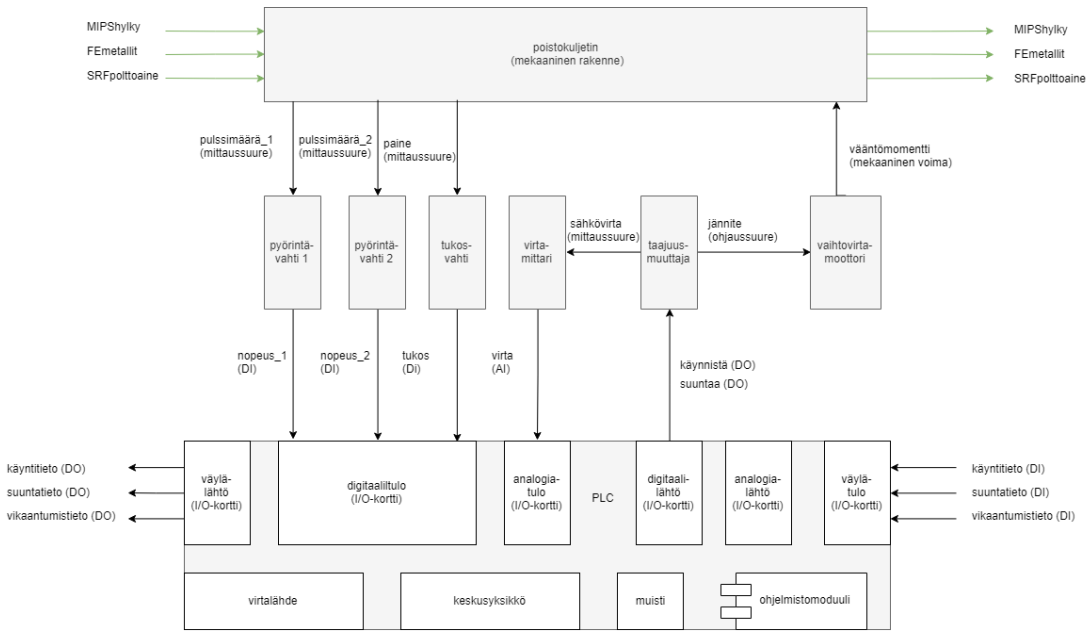
ja analogisiin kortteihin sekä väyläkortteihin. I/O-kortit voidaan edelleen jakaa tulokortteihin ja lähtökortteihin.

Digitaalisissa I/O-korteilla voidaan tallentaa vain kaksi tilaa (1 tai 0). Digitaalisia tulokortteja käytetään esimerkiksi rajakytkimien ja painikekytkimien mittaustiedonkeruussa. Digitaalisia lähtökortteja taas voidaan käyttää moottorin turvatoimintojen ohjaustietojen keruussa.

Analogisilla I/O-korteilla tulot ja lähdöt voidaan tallentaa, esimerkiksi, virtaviestinä (4–20 milliampeeria). Näin voidaan lähettää jatkuva-aikaisia signaaleja. Virtaviesti voidaan skaalata merkitykselliseksi mittaustulokseksi PLC-ohjelmistossa, ohjelmoimalla tarvittavan muunnoksen toteuttava ohjelmistomoduuli.

Väyläkortit ovat digitaalisten ja analogisten I/O-korttien erityistapauksia. Ne on tarkoitettu, kenttälaitteiden sijaan, PLC-laitteiden väliseen kommunikointiin. Väyläratkaisuna voidaan käyttää esimerkiksi Ethernet-verkkoa.

Kuvassa 28 on esitetty poistokuljettimen, matalimman abstraktiotason, yksityiskohtainen malli. Malli sisältää mekaanisen laitteen ja kenttälaitteet (sähköautomaatiolaitteet) sekä PLC-laitteen moduuleineen. Kenttälaitteiden mittaus- ja ohjaussignaalit on kohdistettu analogisiin ja digitaalisiin I/O-kortteihin. väyläsignaalit on kohdistettu vastaaviin väyläkortteihin. Malli sisältää myös anturien mittaussuureet ja toimilaitteen tuottama säädetty suure.




***Kuva 28.*** *Poistokuljettimen yksityiskohtainen kuvaus.*

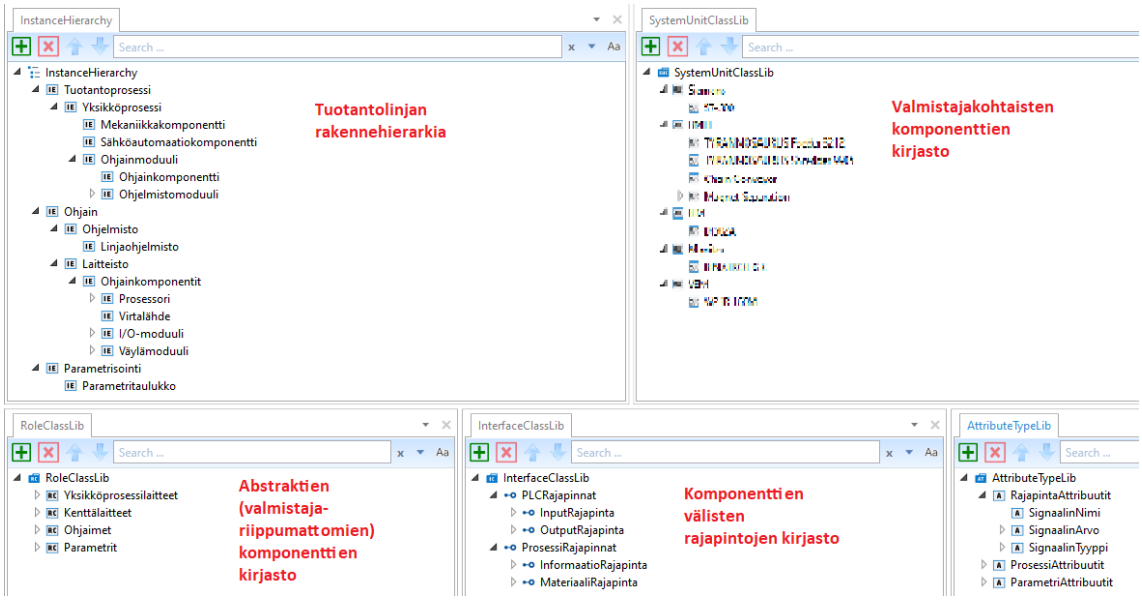
Tuotantolinjan systeemikuvaus voidaan esittää myös AutomationML:n avulla. AutomationML esiteltiin luvussa 3.2.3. Tuotantolinjan AutomationML-kuvaus on esitetty kuvassa

28. Tuotantolinjaan liittyvien valmistajakohtaisten tuotteiden kuvaus on sumennettu immateriaalioikeuksien suojelemiseksi.

AutomationML-kuvauksessa, tuotantolinja on jaoteltu ohjattavaan prosessiin ja prosessia hallitsevaan ohjaimeen. Ohjain jakautuu edelleen ohjainlaitteistoon ja tuotantolinjan ohjausohjelmistoon. Tuotantolinjan kuvaus sopii siis sekä modulaarisille ohjelmistoille että koko linjaa ohjaaville ei-modulaarisille ohjelmistoille.

Myös rajapintojen kuvaus on jaoteltu PLC-ohjaimen rajapintoihin ja ohjattavan prosessin rajapintoihin. Tuotantolinjan parametrit voivat liittyä PLC-ohjelmistoon, ohjelmoitaviin antureihin tai taajuusmuuttajiin. Esimerkki antureiden ja taajuusmuuttajan parametrien kuvauksesta on esitetty liitteessä C.

Yksikköprosessin kuvaus on modulaarinen. Yksikköprosessi koostuu mekaanisesta laitteesta, kenttälaitteista ja ohjelmistomoduuleista sekä ohjainlaitteiston komponenteista. Ohjainkomponentti voi olla esimerkiksi hajautettu I/O-yksikkö. Myös ohjelmistomoduuli jakautuu edelleen pienempiin yksiköihin eli ohjelmistokomponentteihin. Ohjelmistokomponentti voi olla esimerkiksi PLC-funktiolohko.




***Kuva 29.*** *Tuotantolinjan AutomationML-tietorakenne.*

Poistokuljettimen kuljettaessa materiaalia, syntyy kitkavoimia sekä ketjun ja kuljettavan materiaalin massasta aiheutuvia voimia. Moottorin on tuotettava näitä voimia vastaava vääntömomentti, jotta kuljetin liikkuu halutulla nopeudella. Poistokuljettimen säädettävä suure on siis moottorin tuottama vääntömomentti.

Taajuusmuuttajan käyttämä ohjausmenetelmä on suora momenttisäätö (engl. direct torque control), jossa säädettäviä suureita ovat vaihtovirtamoottorin vääntömomentti ja

magneettivuo. Moottorin vääntömomentti ja magneettivuo estimoidaan jännite- ja sähkövirtamittausten perusteella, käyttäen adaptiivista moottorimallia. Moottorimalli siis määrittyy taajuusmuuttajalle asetettujen parametrien mukaisesti.

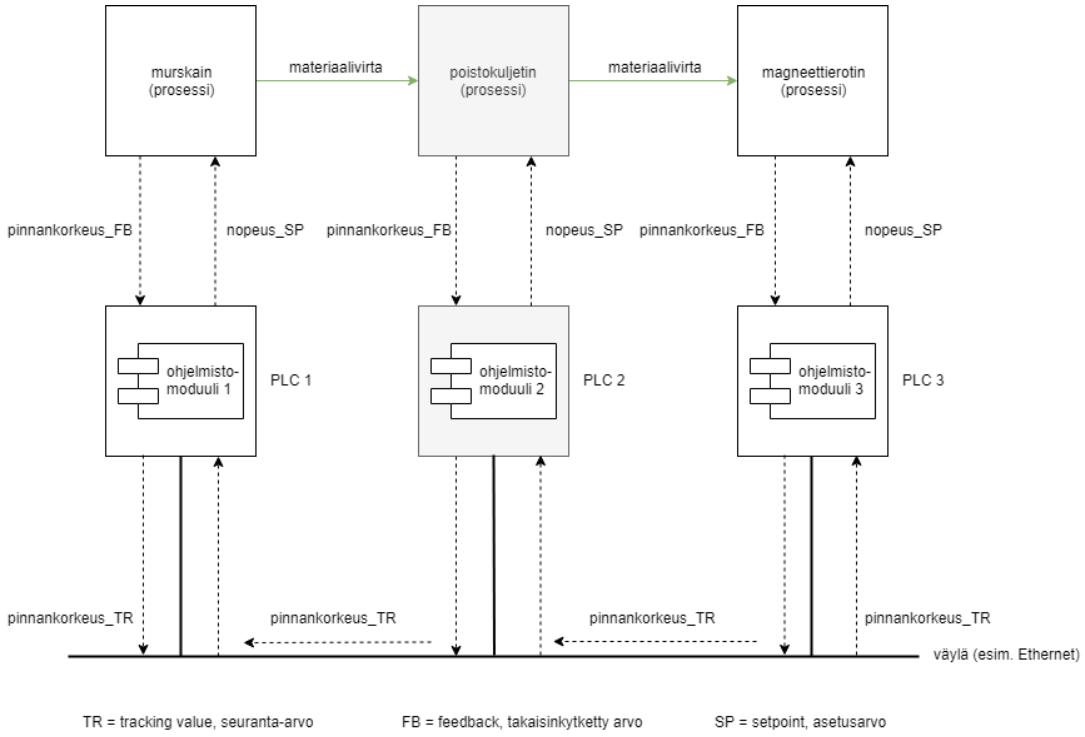
Samalla moottorimallilla estimoidaan myös vaihtovirtamoottorin pyörimisnopeus. Nopeuden säätö tapahtuu PID-säätimellä (engl. Proportional Integral Derivative, PID), jonka asetusarvona on tavoitenopeus ja takaisinkytkentänä estimoitu moottorin nopeus. Kuten edellä todettiin, nopeuden asetusarvoa voidaan vaihtaa logiikkaohjaimella.

Tutkittavan tuotantolinjan PLC-ohjelmistossa, poistokuljettimelle on määritelty kaksi vakionopeutta. Yksi nopeus eteenpäin ajoon ja toinen taaksepäin ajoon. Vakionopeuksia olisi kuitenkin mahdollista ottaa käyttöön useampiakin. Logiikkaohjaimen ja taajuusmuuttajan välinen rajapinta mahdollistaa yhteensä kahdeksan eri nopeuden käyttöönoton. Tällöin sekä eteen- että taaksepäin ajoon olisi valittavissa neljä eri nopeutta, ja nopeutta olisi mahdollista säätää dynaamisesti.

Dynaaminen nopeuden säätö voidaan toteuttaa luvussa 4.2.1 kuvatun imuohjauksen mukaisesti. Kuljettimien nopeutta säädetään siis polttoaineen tarpeeseen perustuen. Nopeuden optimoinnilla voidaan minimoida prosessilaitteiden energiankulutus, energiataloudellisten ajonopeuksien avulla on mahdollista saavuttaa huomattavia taloudellisia hyötyjä.

Polttoainevaraston pinnankorkeuden säätö on regulointitehtävä, jossa pinta pyritään pitämään halutulla tasolla. Polttoainevaraston pinnankorkeuden seuranta-arvo voidaan määrittää tavoitetason ja mitatun polttoainetason välisenä erotuksena. Taajuusmuuttajakäytöllä, nopeuden asetusarvolle voitaisiin antaa neljä ennalta määrättyä arvoa kullekin kulkusuunnalle. Nopeuden säätö on siis servotehtävä, jossa seurataan muuttuvaa nopeuden asetusarvoa. Kussakin yksikköprosessissa, asetusarvo valitaan polttoaineentarpeen seuranta-arvon ja yksikköprosessilta mitatun materiaalikertymän perusteella. Materiaalikertymiä voidaan mitata esimerkiksi rajakytkimillä tai optisilla antureilla.

Tuotannonohjauksen ohjauspisteenä on tuotantolinjan pullonkaula eli murskain. Tällöin ohjaukset voidaan ajoittaa kuljettimien kulkuviiveiden mukaan. Pullonkaula tulee pitää jatkuvasti kuormitettuna. Askelsyötin syöttää murskaimelle raaka-ainetta murskaimen pinnankorkeuteen perustuen. Polttoaineen tarpeen ollessa alhainen, voidaan kuljettimien ja murskaimen nopeuksia laskea taloudellisimmalle tasolle. Askelsyöttimen ja polttoainevaraston puskurit mahdollistavat polttoaineen kysynnän vaihtelun tasaamisen. Näin ollen vältetään tilanne, jossa polttoaineen kysyntä ylittäisi tarjonnan. Kuvassa 30 on esitetty dynaamista nopeuden säätöä kuvaava alustariippuva malli.




***Kuva 30.*** *Dynaamisen nopeuden säädön alustariippuva malli.*

### 4.2.3 Informaatiorajapintojen kuvaus

Rajapintasopimusten tekeminen on tärkeä vaihe systeemin mallinnuksessa. Modulaarisuus vaatii hyvin määriteltyjä ja dokumentoituja rajapintoja. Tällöin moduulien käyttäminen on mahdollista erilaisissa systeemikonfiguraatioissa. Rajapintasopimukset mahdollistavat myös sen, että useampi ohjelmoija voi kehittää moduuleja rinnakkain. Tällöin moduulin kehittäjän ei tarvitse tuntea muiden moduulien toteutusta, vaan moduulien tarjoamien ja vaatimien rajapintojen tunteminen riittää. Nämä periaatteet pätevät sekä simulointimallien moduulien että PLC-ohjelmiston moduulien kehittämiseen.

Vakiintuneet rajapinnat helpottavat moduulien integrointia osaksi laajempaa systeemiä. Rajapintoja muutettaessa tulee huomioida muutosten vaikutukset koko systeemiin. Rajapinnan muutos vaatii aina vähintään kahden moduulin toteutuksen muuttamista (rajapinnan lähdemoduuli ja kohdemoduuli). Tällöin myös moduulien aiemmat versiot ovat yhteensopimattomia uusia rajapintoja käyttäviin konfiguraatioihin, eivätkä rajapintamuutokset siten ole taaksepäin yhteensopivia.

Rajapintojen abstraktiotasot helpottavat muutosten vaikutusten tunnistamista. Luvussa 3.2.2 esiteltiin rajapintojen abstraktiotaso, joka perustuu prosessimallin määrittelemiin prosessisignaaleihin. Tässä tutkimuksessa on käytetty vastaavanlaista abstraktiotasoa,

joka kuvaa rajapintasignaalit kenttälaitetasolla. Rajapinnan signaalit ovat siis antureiden tarjoamia ja toimilaitteiden vaatimia signaaleja. Näin rajapinnat voidaan valita systemaattisesti reaalimaailman tuotantolinjakonfiguraation perustuen. Kenttälaitetason abstraktiosignaalien avulla on helppo huomata, mikäli uusien ohjelmistomoduulien toiminnallisuudet tarjoavat uusia signaaleja toimilaitteille. Vastaavasti, uudet toiminnallisuudet voivat vaatia tietyn signaalin mittaamista anturilla tai estimointia tilahavaitsijalla. Rajapintojen abstraktiotaso helpottaa myös ohjelmistomoduulien testaamista. Rajapintoja tarkastelemalla voidaan valita testattavaksi vain ne ohjelmistomoduulit, joihin muutokset vaikuttavat.

Antureiden analogisten lähettimien signaalit voidaan esittää virtaviesteinä, jotka anturille kehitetty ohjelmistomoduuli skaalaa merkitykselliseksi mittaustulokseksi. Rajakytkimien ulostulosignaalit voidaan esittää totuusarvona siten, että arvo 1 kuvaa normaalin operoinnin sallivaa eli turvallista tilaa ja arvo 0 poikkeavaa tilaa, kuten vikaantumista.

Toimilaitteiden ohjauspyynnöt voidaan esittää niille asetettujen rajoitusten perusteella. Esimerkiksi, moottorin syöttöjännitteen arvoväli voidaan määritellä suurimman sallitun jännitteen itseisarvona. Moottorin turvatoimintoihin liittyvät ohjauspyynnöt voidaan esittää totuusarvona vastaavasti kuin edellä kuvattujen antureiden tapauksessa. Esimerkiksi, moottorin käynnistäminen voidaan kuvata arvolla 1 ja pysäyttäminen arvolla 0.

Signaalien nimeämiskäytäntöjen määrittely on systemaattinen lähestymistapa rajapintasignaalien yksilöintiin. Merkityksellisesti ja määrämuotoisesti nimetyt informaatiosignaalit edistävät ohjelmistojen ja simulointimallien uudelleenkäyttöä sekä mahdollistavat ohjelmistojen ja simulointimallien kehitykseen liittyvien vaiheiden ainakin osittaisen automatisoinnin.

Energiateollisuudessa laajalti käytetty nimeämiskäytäntö perustuu Kraftwerk Kennzeichen System –tunnistejärjestelmään (voimalaitosten tunnistejärjestelmä, KKS). Laitoksille, osasysteemeille sekä niiden laitteille ja komponenteille voidaan määritellä yksilöllinen KKS-tunniste. KKS-tunnisteita voidaan soveltaa myös automaatiotoimintojen, säätöpiirien ja signaalien nimeämiseen.

KKS-tunniste jakautuu neljään erittelytasoon (engl. breakdown level), jotka ovat laitos-, toiminto-, säätöpiiri- ja signaalitaso. KKS-tunnisteen tasot ja niiden selitykset sekä polttoainevaraston pinnankorkeuden mittausta kuvaava esimerkkitunniste on esitetty taulukossa 4. Lisätietoa KKS-tunnisteen soveltamisesta löytyy lähteestä [54].

***Taulukko 4.*** *KKS-tunnisteen rakenne.*

| **Tason numero** | **0** | **1** | **2** | **3** |
|---|---|---|---|---|
| **Tason nimi** | Laitos | Toiminto | Säätöpiiri | Signaali |
| **Esimerkki-tunniste** | 1 | EAE12 | CL001 | XQ01 |

KKS-tunnistetta käytetään siis voimalaitoksen rakenteen erittelemiseksi pienempiin osakokonaisuuksiin. Tunnisteet koostuvat numeroista ja kirjaimista. Tunnisteet ovat laitoskohtaisia, ja tavallisesti laitossuunnittelijat määrittelevät ne. Standardoidun KKS-tunnisteen perusteella, signaalit voidaan yhdistää niitä vastaaviin automaatiotoimintoihin yksikäsitteisesti.

Tapaustutkimuksen tuotantolinjalle ei kuitenkaan ole määritelty KKS-signaalitunnisteita. Tässä tutkimuksessa, signaalien yksilöintiä havainnollistetaan KKS-tunnistetta yksinkertaisemmalla TSNC-tunnisteella (engl. Tagname Signal Naming Convention, TSNC). TSNC-tunnisteet soveltuvat esimerkiksi PLC-ohjelmistojen signaalien nimeämiseen. TSNC-nimeämiskäytännöstä löytyy lisätietoa lähteestä [55].

KKS-tunnisteen kaltaisesti, TSNC-tunniste koostuu 15 merkistä. Merkit voivat olla numeroita tai kirjaimia. Myös TSNC-tunniste on jaoteltu neljään eri tasoon, taulukon 5 mukaisesti.

***Taulukko 5.*** TSNC*-tunnisteen rakenne.*

| **Tason numero** | **0** | **1** | **2** | **3** |
|---|---|---|---|---|
| **Tason nimi** | Sijainti | Prosessilaite | Komponentti | Signaali |
| **Esimerkki-tunniste** | WPL | DCR1 | IT | XI1 |
| **Selite** | Waste Processing Line | Discharge Conveyor | Current Transmitter | Current Indication |

Sijaintitaso kuvaa tietyn systeemin fyysistä sijaintia laitoksella kolmen merkin lyhenteellä. Taulukon 5 esimerkissä, sijainnilla on kuvattu voimalaitoksen jätteenkäsittelylinjastoa (engl. Waste Processing Line). Prosessilaitetasolla yksilöidään fyysinen prosessilaite, kuten poistokuljetin (engl. Discharge Conveyor), neljän merkin avulla. Komponenttitaso määrittelee tunnisteen prosessilaitteen yksittäiselle komponentille, esimerkiksi virtamittarin lähettimelle (engl. current transmitter). Komponentin nimeämiseen on käytössä kaksi merkkiä, jotka voivat olla numeroita tai kirjaimia. Signaalitason tunniste määrittelee signaalin suunnan ja merkityksen. Tässä tutkimuksessa signaalitunnisteen ensimmäinen

kirjain on X, Y tai Z. Kirjain X kuvaa osasysteemiin liittyvää mittaustietoa eli säätimen tulosignaalia, Y osasysteemin ohjausta eli säätimen lähtösignaalia ja Z osasysteemien välistä tulo-/lähtösignaalia eli säätimeltä toiselle välitettäviä tilatietoja.

TSNC-tunniste koostuu siis kirjainlyhenteistä, jotka ovat vapaasti valittavissa kullakin systeemin erittelytasolla. TSNC- tunniste on siis KKS- tunnistetta intuitiivisempi. Vapaavalintaisuus ja standardoitujen lyhenteiden puuttuminen ovat kuitenkin TSNC-tunnisteen heikkouksia, sillä ne lisäävät tunnisteiden monitulkintaisuutta. Signaalien nimeämiseen kannattaa siis käyttää KKS-tunnisteita, mikäli ne ovat saatavilla.

Kuten luvussa 3.2.2 todettiin, signaalille tulee määritellä yksilöllisen nimen lisäksi arvoväli ja tyyppi. Signaali voi siis saada arvoja minimi- ja maksimiarvojen väliltä. Tulo- ja lähtösignaaleille tulee määritellä sallitut lukuarvot ja käytetyt mittayksiköt. Signaali on tyypiltään joko analogiasignaali tai digitaalisignaali. Analogiasignaalit ovat jatkuva-arvoisia signaaleja ja digitaaliset diskreettejä signaaleja. Tässä työssä käsiteltävät digitaalisignaalit ovat binäärisignaaleja, joilla on vain kaksi tilaa (1 ja 0).

Logiikkaohjaimien I/O-kortit (AI, AO, DI tai DO) määrittelevät tulo- ja lähtösignaalien tyypit. I/O-kortieissa on useita I/O-portteja. Signaalit kohdistetaan I/O-portteihin muistipaikkoja kuvaavien osoitteiden avulla. Ohjelmistotasolla I/O-signaalit ovat siis muuttujia, joille on määriteltävä absoluuttinen ja symbolinen osoite sekä tietotyyppi. Absoluuttinen osoite kertoo mihin muistipaikkaan ja bittiin muuttuja on tallennettu. Symbolinen osoite mahdollistaa symbolisen nimen (kuten KKS-tunniste tai TSNC- tunniste) määrittelemisen absoluuttisille osoitteille. Symbolisten osoitteiden käyttö yksinkertaistaa ohjelmointia sekä helpottaa ohjelmiston ymmärrettävyyttä ja testausta. Tietotyyppi määrittelee muuttujan ominaisuudet ja käyttötavat. Käytännössä tietotyyppi siis kuvaa muuttujan esitystavan ja sallitut arvot. Tyypillisiä tietotyyppejä ovat real, integer ja boolean. Tietotyyppi real kuvaa liukulukua, integer kokonaislukua ja boolean totuusarvoa.

Rajapinnat ovat ihmisten välisiä sopimuksia. Valittujen rajapintojen tulisi olla kaikkien kehitystyöhön osallistuvien osapuolten ymmärrettävissä. Tähän tarkoitukseen soveltuu selkeät rajapintakuvaukset. Liitteessä B on esitetty taulukkopohjainen rajapintakuvaus poistokuljettimen alustariippuvalle mallille.

Ohjelmistomoduulien I/O-signaalit voidaan valita kenttälaitetason I/O-signaaleita kuvaavan abstraktiotason mukaisesti. Tällöin rajapinnat perustuvat, mielivaltaisten sopimusten sijaan, tuotantolinjan automaatiotoimintoihin ja todelliseen laitteistoon.

Tätä systemaattista lähestymistapaa on havainnollistettu taulukoissa 6 ja 7. Taulukon 6 tiedot kuvaavat kuvan 27 informaatiomuuttujia. Ohjausten nimien tulisi olla verbejä, jotka kuvaavat pyydettyä ohjaustoimintoa. Tilojen nimet tulisi olla adjektiiveja tai substantiiveja, jotka kuvaavat tilojen merkitystä.

***Taulukko 6.*** *Kenttälaitetason I/O-signaalien ominaisuudet.*

| Signaalin nimi | I/O-tyyppi | Minimi-arvo | Maksimi-arvo | Kuvaus |
|---|---|---|---|---|
| virta | AI | 4 | 20 | virtamittarin lähettimen virta-arvo virtaviestinä (4…20 mA) |
| nopeus_1 | DI | 0 | 1 | pyörintävahdin 1 nopeuden tilatieto (1 = riittävä nopeus, 0 = alinopeus) |
| nopeus_2 | DI | 0 | 1 | pyörintävahdin 2 nopeuden tilatieto (1 = riittävä nopeus, 0 = alinopeus) |
| ei-tukosta | DI | 0 | 1 | tukosvahdin tilatieto (1 = ei-tukosta, 0 = tukos havaittu) |
| käynnistä | DO | 0 | 1 | asettaa moottorin päälle tai pois päältä (1 = päälle, 0 = pois) |
| suuntaa | DO | 0 | 1 | asettaa liikkeen suunnan (1 = eteen, 0 = taakse) |

Standardin IEC 61131-3 mukaan ohjelmistotason I/O-muuttujat ovat PLC-ohjelmistomoduulien parametreja. Useiden I/O-muuttujien hallinnointiin voidaan käyttää PLC-signaalitaulukoita (engl. PLC tag table). Hyvien ohjelmointitapojen mukaisesti, PLC-signaalitaulukkoon tulee lisätä myös kommentti. Kommentti on signaalin toiminnallisuutta kuvaava selkokielinen teksti. Kommentti siis määrittelee merkityksellisen nimityksen (engl. designation) signaalitunnisteelle (symbolinen nimi). Taulukossa 7 on esitetty (taulukon 6) informaatiosignaalien abstraktiotasoja vastaavat PLC-ohjelmiston I/O-signaalit.

***Taulukko 7.*** *Ohjelmistomoduulin I/O-signaalien ominaisuudet.*

| Symbolinen nimi | Tieto-tyyppi | Osoite | Kommentti |
|---|---|---|---|
| FPS_DCVR_IT_XI1 | Real | %I0.1 | Discharge Conveyor Current Trasmitter Indication |
| FPS_DCVR_S1_XS1 | Bool | %I0.2 | Discharge Conveyor Speed Switch 1 Status |
| FPS_DCVR_S2_XS2 | Bool | %I0.3 | Discharge Conveyor Speed Switch 2 Status |
| FPS_DCVR_S3_XF1 | Bool | %I0.4 | Discharge Conveyor Chocked Flow Switch Status |
| FPS_DCVR_M1_YC1 | Bool | %Q0.1 | Discharge Conveyor Motor Control ON/OFF Command |
| FPS_DCVR_M1_YC2 | Bool | %Q0.2 | Discharge Conveyor Motor Control FW/BW Command |

Tutkittavan yrityksen tapauksessa, PLC-ohjelmistojen kehitystyössä käytetään Siemensin TIA Portal (Step 7) –ohjelmointityökalua. Työkalu mahdollistaa PLC-laitteistokonfiguraatioon liittyvän tiedonvaihdon AutomationML-merkintäkielellä muiden suunnittelutyökalujen, kuten Eplan, kanssa. Yrityksessä käytössä oleva Eplan on sähköautomaation suunnittelutyökalu, jolla luodaan piirikaavioita.

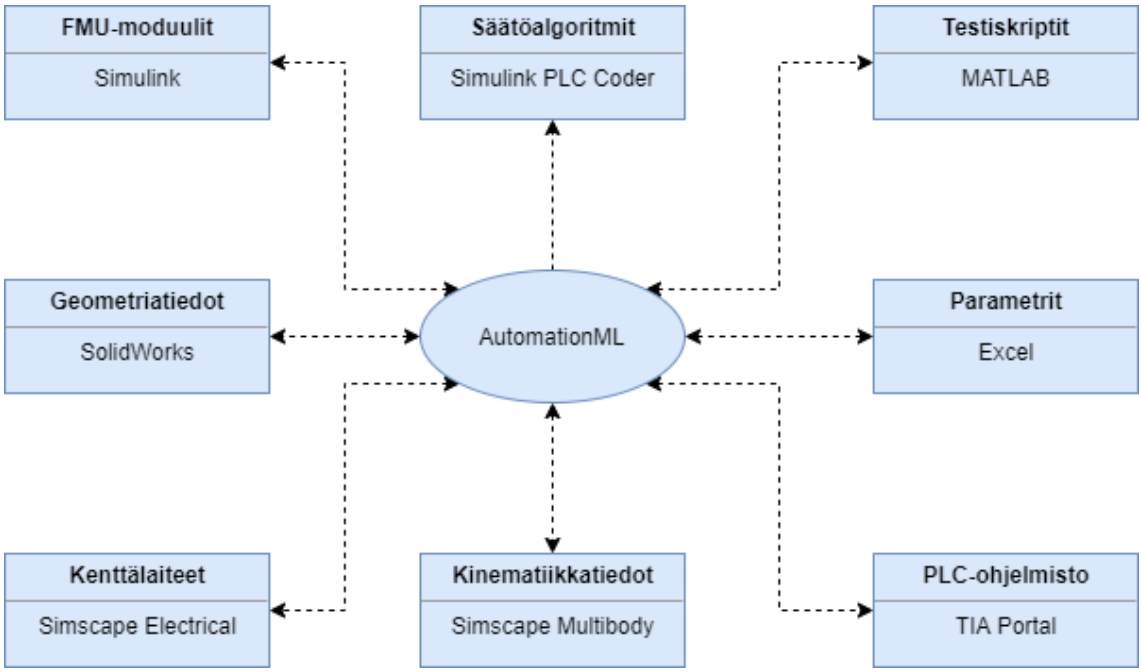
AutomationML- tiedostojen avulla, Eplanissa määritellyt PLC-keskusyksiköt ja niihin liittyvät I/O-laitteet sekä I/O-signaalitaulukot voidaan tuoda TIA Portaliin. Näin voidaan säästää ohjausjärjestelmien suunnitteluun käytettyä aikaa, lisätä uudelleenkäytettävyyttä sekä vähentää manuaalisen kopiointityön aiheuttamien virheiden määrää. Huomionarvoista kuitenkin on, että vain TIA Portalin kanssa yhteensopivien tuotteiden laitteistokonfiguraatiotiedot voidaan tuoda Eplanista TIA Portaliin. Eplanin ja TIA Portalin välisen tiedonvaihdon toteutuksesta löytyy lisätietoa lähteestä [56].

TIA Portal ei kuitenkaan tue AutomationML-merkintäkielen käyttämää PLCopen standardia ohjaussovellusten koodin tallentamiseen, vaan hyödyntää omaa TIA Openess –ohjelmointirajapintaa (engl. application programming interface). Rajapinnan avulla, TIA Portal -ohjelmointiympäristöön voidaan tuoda muilla työkaluilla, kuten Simulink PLC Coderilla, generoituja Step 7 -yhteensopivia ohjelmistomoduuleja.

TIA Openess käyttää XML-tiedostoja työkalujen välisessä tiedonvaihdossa. Step 7 –ohjaussovelluksen lisääminen AutomationML-tietorakenteeseen vaatisi kuvauksen laajentamista, ulkopuolisella XML-formaattilla. Simscape Multibody ja SolidWorks eivät myöskään hyödynnä AutomationML-formaatin COLLADA-standardia, vaan STL-tiedostoformaattia (engl. stereolithography). Myös geometria- ja kinematiikkatietojen integrointi vaatisi lisää implementointityötä. Kuten luvussa 3.2.3 todettiin, Simulink tukee FMU-moduuleja. FMU-moduulien lisääminen AutomationML-kuvaukseen vaatisi myös integrointityötä, sillä AutomationML ei sellaisenaan tue FMU-rajapintastandardia.

Suunnittelutyökalujen AutomationML-tuki on siis toistaiseksi puutteellinen saumattoman työkaluketjun toteuttamiseksi. CAD-työkalu (SolidWorks) ei kommunikoi sähköautomaation suunnittelutyökalun (Eplan) kanssa. Edestakainen tiedonvaihto PLC-ohjelmointityökalun (TIA Portal) ja simulointityökalun (Simulink) välillä ei myöskään onnistu ilman import/export-työkalun implementointia.

Tiedonvaihto eri suunnittelutyökalujen välillä muodostaa pullonkaulan dynaamisen simulointimallin kehitysprosessissa. Kuvassa 31 on havainnollistettu suunnittelutyökalujen tiedonvaihdon lähestymistapaa, jossa AutomationML-tiedostot muodostavat yhteisen tietokannan eri suunnittelutyökaluille.




***Kuva 31.*** *Integroitavat suunnittelutyökalut.*

### 4.2.4 PLC-ohjelmiston modulaarinen rakenne

Tuotantolinjan modulaarinen rakenne on edellytys ohjelmistojen systemaattiselle uudelleenkäytölle. Tapaustutkimuksen tuotantolinjan PLC-ohjelmisto ei kuitenkaan ole modulaarinen. Ohjelmistojen uudelleenkäyttö perustuu, moduulien sijaan, vanhojen projektitoteutusten kopiontiin. Tällöin vanhat projektit toimivat koodipohjina, joita muokataan seuraavassa ohjelmistokehitysprojektissa. Tämä menettely vaikeuttaa ohjelmistojen ylläpidettävyyttä sekä varianttien ja versioiden hallintaa.

SRF-tuotantolinjan automaatiolaitteita, kuten logiikkaohjaimia, antureita ja moottoreita, on modernisoitava laitoksen elinkaaren aikana. Myös ohjelmiston toiminnallisuudet vaativat mukauttamista, esimerkiksi, laitteistotukipäivityksistä tai suorituskykypäivityksistä johtuen. Tuotantolinja vaatii siis modulaarisen ohjelmiston, joka helpottaa ylläpitoa, testausta ja uudelleenkonfigurointia.

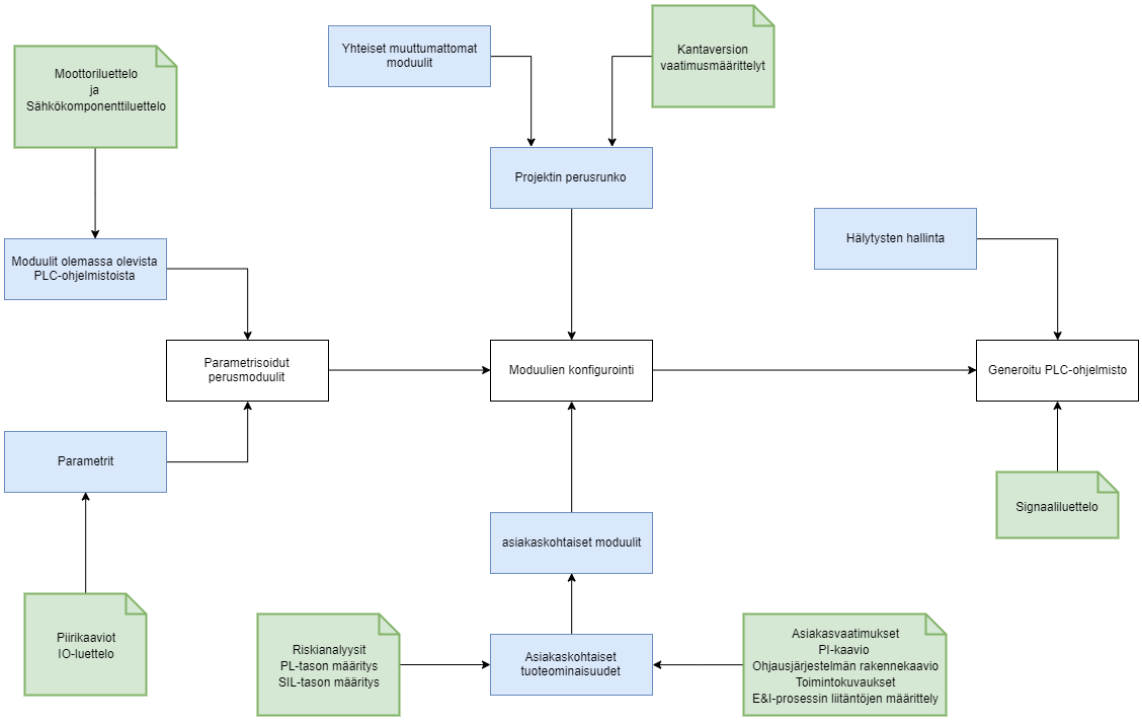
PLC-ohjelmistoissa, ohjelmistomoduuli voidaan rakentaa funktiolohkon (engl. function block) avulla. Funktiolohkolla toteutetaan tietty ohjelmiston toiminnallisuus, kuten monitorointi- tai ohjausalgoritmi, minkä jälkeen moduulia voidaan käyttää uudelleen eri ohjelmistoissa. Moduulit ovat siis ohjelmiston rakennuspalikoita, joita yhdistelemällä voidaan konfiguroida erilaisia ohjelmistoja.

Modulaarisen ohjelmiston konfiguroinnissa voidaan käyttää kahta vastakkaista lähestymistapaa, jotka perustuvat kantaversion määrittelemiin ohjelmistoprojekteihin. Ensimmäisessä vaihtoehdossa, ohjelmistoprojekti sisältää tuotantolinjan tai prosessilaitteen

kaikki toiminnallisuudet. Tästä maksimikonfiguraatiosta riisutaan tai piilotetaan tarpeettomat toiminnallisuudet asiakastarpeita vastaaviksi. Moduulien poistaminen käytöstä saattaa kuitenkin aiheuttaa ongelmia, moduulien välisten vahvojen riippuvuuksien vuoksi.

Esimerkiksi turva-automaatiotoimintoihin liittyvät lukitusketjut ja turva-alueet muodostavat laajoja ja kompleksisia rajapintoja. Tällöin yksittäisen moduulin eliminointi saattaa vaatia tietyn rajapinnan muuttamista luotettavuuden ja vikasietoisuuden sekä turvallisen toiminnan takaamiseksi. Ohjelmistoon vaadittavat muutokset ovat näin ollen työläitä ja haastavia toteuttaa. Tässä tutkimuksessa suositellaankin vaihtoehtoista lähestymistapaa.

Toisessa vaihtoehdossa ohjelmistoprojektille luodaan runko, joka sisältää vain kaikille ohjelmistoille yhteiset perustoiminnallisuudet. Tähän perusohjelmistoprojektiin lisätään toiminnallisuuksia yhdistelemällä jo olemassa olevia ja asiakaskohtaisesti kehitettäviä moduuleita. Näin voidaan konfiguroida useita erilaisia ohjelmistoja saman kantaversion pohjalta. Kantaversion määrittelemä PLC-ohjelmiston runko pysyy identtisenä kaikille uusille projekteille. Tätä lähestymistapaa on havainnollistettu kuvassa 32.




***Kuva 32.*** *Modulaarisen PLC-ohjelmiston konfigurointi.*

Automaatio-ohjelmistot ovat vahvasti alustariippuvaisia, mikä vaikeuttaa ohjelmistojen modulaarisen rakenteen implementointia. Modulaarisen ohjelmoinnin soveltaminen vaatii siis alustan standardointia laitteisto- ja ohjelmistotasolla. Tällöin määritellyille laitteistokomponenteille (kuten anturit ja toimilaitteet) voidaan rakentaa toiminnalliseen erotte-

luun perustuen uudelleenkäytettäviä ohjelmistomoduuleja. Yksittäisen ohjelmistomoduulin tehtävänä on siis toteuttaa tietty toiminnallisuus, kuten moottorin pyörimisnopeuden säätö tai anturin mittaustiedon prosessointi.

Käytännössä alustan standardisointimahdollisuuksia rajoittavat projektien kertaluontoisuus sekä kustannus- ja aikataulupaineet. Voimalaitosten laitetoimitusprojektit ovat ainutkertaisia kokonaisuuksia. SRF-tuotantolinjaan liittyvien prosessilaitteiden rakenteissa ja toiminnassa on siis projektikohtaisia eroja, ja erilaiset toteutukset vaativat varioitavia ohjelmistomoduuleja.

Ohjelmistomoduulit koostuvat yhdestä tai useammasta ohjelmistokomponentista. Tutkittavaa tuotantolinjaa ohjataan Siemensin Step 7 PLC-ohjelmistolla. Siemensin tukemia PLC-ohjelmistokomponentteja ovat tietolohkot ja käyttäjän määrittelemät tietotyypit sekä funktiolohkot.

Ohjelmiston muuttujat ovat joko kaikille ohjelmistokomponenteille yhteisiä globaaleja muuttujia tai tiettyyn funktiolohkoon liittyviä muuttujia. Tietolohkot (engl. data block) mahdollistavat muuttujien uudelleenkäytön ohjelmistossa. Muuttujat tallennetaan tietolohkoihin, joiden rakenne on vapaasti valittavissa. Valittu tietorakenne luodaan käyttäjän määrittelemien tietotyyppien avulla. Tietotyypit mahdollistavat useista muuttujista koostuvien rajapintojen kapseloinnin yhdeksi muuttujaksi. Rajapintojen kapselointi helpottaa modulaarista ohjelmointia.

Kuvassa 32 esitettyjen parametrisoitavien perusmoduulien tehtävänä on määritellä yksittäisen toimilaitteen tai anturin toiminta. Olio-ohjelmoinnin periaatteiden mukaisesti, funktiolohkot voidaan ajatella luokkina, joiden ominaisuudet määräytyvät funktiolohkoon liittyvien muuttujien perusteella. Kenttälaitevariantteja kuvaavat muuttujat voidaan siis määritellä funktiolohkolle syötettävinä parametreina. Parametrit voivat olla esimerkiksi ohjelmistomoduulien rajapintoja kuvaavia muuttujia.

Prosessilaitteille laaditut moottori- ja sähkökomponenttiluettelot määrittelevät kenttälaitteet, joten projektin perusmoduulit voidaan valita näiden dokumenttien perusteella. Olemassa olevista moduuleista laaditaan valmiita moduulikirjastoja, joista ohjelmistokehittäjät voivat tuoda tarvittavat perusmoduulit uuteen ohjelmistoprojektiin. Moduulikirjastot nopeuttavat ohjelmointityötä ja yhdenmukaistavat ohjelmistojen toteutuksia.

Moduuleihin liittyvät parametrit määräytyvät piirikaavioiden perusteella laadituista IO-luetteloista. Parametrit voidaan kuvata esimerkiksi Excel-työkalulla, jonka jälkeen parametritaulukko tuodaan ohjelmointiympäristöön. Edellisen luvun taulukossa 7 on esitetty esimerkki parametritaulukosta.

Tuotantolinjan modulaarisen tuotetietokuvauksen tavoitteena on mahdollistaa asiakkaalle tarjottavan ratkaisun konfigurointi valmiista tuotteista eli tiettyyn yksikköprosessiin liittyvistä laitteistosta ja ohjelmistosta. Ohjelmisto- ja laitteistokonfiguraatioista voidaan

muodostaa tuoteperheitä prosessilaitteiden (esim. jätemurskaimet tai biomurskaimet) tai laitostyyppien (esim. jätettä tai biomassaa polttavat lämpövoimalaitokset) mukaisesti.
Mikäli asiakas vaatii muutoksia tarjolla oleviin tuotteisiin, projektissa kehitetään myös asiakaskohtaisia moduuleja. Muutokset voivat olla esimerkiksi uudentyyppisiä laitteistokomponentteja tai uusia toimintoja ohjelmistoihin. Asiakasvaatimukset voidaan kuvata asiakaskohtaisina tuoteominaisuuksina. Uudet tuoteominaisuudet tulee määritellä prosessilaitteelle tuotettavassa dokumentaatiossa. Muutokset vaikuttavat muun muassa luvussa 3.3.1 määriteltyihin dokumentteihin. Muutosten dokumentointi ja implementointi vie tyypillisesti paljon aikaa ja lisää näin projektin kustannuksia.
Rajapintasignaalien abstraktiotasot helpottavat muutosten hallintaa. Rajapintakuvausten avulla voidaan tarkistaa, että tarvittavat moduulit on lisätty ohjelmistokonfiguraatioon. Lisäksi ohjelmiston konfiguroinnissa on huomioitava hälytysten hallinta. Vikaantumistilanteet ja muutokset ohjaustavassa, esimerkiksi automaattiohjauksesta manuaaliohjaukseen, aiheuttavat hälytyksiä, jotka on välitettävä operaattorille. Turvamekanismien toteutuksessa on laitoskohtaisia eroja, joten yksittäiset hälytykset eivät voi sisältyä moduulien toteutuksiin. Hälytykset on siis kerättävä yhteen ja käsiteltävä moduulien ulkopuolella.
Tuotantolinjan laitteiston ja ohjelmiston modulaarinen rakenne helpottaa yksikköprosessikohtaisten toimintojen kehittämistä ja testaamista. Kompleksisten haasteiden ratkaisemisessa (esim. materiaalivirtojen ja energiankulutuksen optimointi sekä kehittyneiden säätöalgoritmien suunnittelu) voidaan hyödyntää simulointia. Simulointimalleilla voidaan tutkia säätöalgoritmin toimivuutta jo ennen PLC-koodin implementointia. Algoritmin implementoinnin jälkeen ohjelmiston mallipohjaiseen testaukseen ja validointiin käytetään HIL-simulointia. Mallipohjaisen suunnittelu siis siirtää ohjelmistokehityksen painopisteen ohjelmointityöstä mallinnukseen. Fyysisen tuotantolinjan ja sen virtuaalisen prototyypin väliset erot aiheutuvat käytettyjen simulointimallien epätäydellisyyksistä. Näin ollen ohjelmistojen parametrit vaativat hienosäätöä käyttöönotossa.

## 4.3 Tuotantolinjan versionhallinta

Tässä luvussa käsitellään ohjelmistojen ja simulointimallien versionhallintaa. Ensimmäisessä alaluvussa määritellään versionhallinnalle asetettavat tavoitteet ja vaatimukset. seuraavassa alaluvussa kartoitetaan versionhallinnan toteutukseen soveltuvia työkaluja. Lopuksi esitellään versioiden hallinnassa käytettäviä menettelyjä.

### 4.3.1 Tavoitteiden ja vaatimusten määrittely

Versionhallinnan tavoitteet ja vaatimukset määriteltiin yhdessä tutkittavan yrityksen tuotekehitysorganisaation kanssa. Tutkimusprojektin alussa määriteltiin versionhallinnan

karkeat tavoitteet, aiheen rajausta koskevan vapaamuotoisen kokouksen yhteydessä. Tavoitteita ja vaatimuksia täsmennettiin tutkimusprojektin puolivälissä. Versionhallinnasta pidettiin tällöin toinen vapaamuotoinen kokous. Kokouksessa muodostettiin selkeä näkemys siitä, miten ohjelmistojen ja niihin liittyvän laitteiston tuotetietoa tulisi hallita yrityksessä.

Versionhallinnan vähimmäistavoitteeksi asetettiin ratkaisu, joka mahdollistaa tällä hetkellä käytössä olevien ohjelmistoversioiden haltuunoton. Eri ohjelmistoversioiden tulisi siis olla selkeästi identifioitavissa. Tallennettujen versioiden metatiedoista tulisi ilmetä mihin projektiin ja laitokseen tietty ohjelmisto liittyy. Ohjelmistot tulisi myös tallentaa siten, että kaikki ohjelmistot ja niiden versiohistoriat ovat löydettävissä yhdellä työkalulla. Versionhallintatyökalu tulisi integroida yhdeksi tietotyökalukokonaisuudeksi dokumentaation hallinnan ja tuotetiedon hallinnan sekä tuotannon ohjauksen työkalujen kanssa.

Edellä mainittujen vähimmäistavoitteiden lisäksi muodostettiin näkemys versionhallinnan pitkän tähtäimen tavoitteista. Ohjelmistojen versionhallinta tulisi kytkeä osaksi, kehityksen alla olevaa, elinkaaren hallinta –ratkaisua. Ratkaisussa tuotantolinjat kuvattaisiin modulaarisina tuoteolioina. Tuoteolion tarkoituksena on kuvailla tiettyyn tuotantolinjaan liittyvä tuotetieto yksityiskohtaisella tasolla. Tuoteolio yhdistäisi prosessilaitteeseen liittyvän mekaanisen laitteiston ja sähköautomaatiolaitteiston sekä ohjelmiston, yhdeksi hallittavissa olevaksi tuotetietokokonaisuudeksi. Myös ohjelmistoihin, antureihin ja taajuusmuuttajiin liittyvät parametrit tulisi linkittää tuoteolion avulla tuotteisiin ja ohjelmistoversioihin. Lähestymistavan keskeisimpiä tavoitteita ovat tuotteisiin liittyvien kustannusten parempi hallinta ja tuotteiden toteutuksien vakiointi. Vakiointi toteutettaisiin perustamalla kantaversioita, jotka kuvailevat tuotteen laitteisto- ja ohjelmistokomponentit.

Asetettujen tavoitteiden pohjalta johdetut toiminnalliset vaatimukset voidaan esittää seuraavasti:

- Tuotteen konfiguraatioiden, moduulien ja komponenttien sekä ohjelmistoversioiden identifiointi
- Toimitettuun tuotteeseen liittyvän laitteiston ja ohjelmiston elinkaaren hallinta
- Tuotteeseen liittyvien muutosten ja kantaversioiden hallinta
- Projektin kustannusten allokointi tuotteille ja niiden komponenteille
- Tuotteeseen liittyvän tilan (kuten työn alla, testattu tai julkaistu) seuranta
- Tuotteen valmiusasteen ja kannattavuuden seuranta
- Tuotetarjoaman määrittely laitteisto-, ohjelmisto- ja parametrikirjastojen avulla.
- Tuotetietorakenteen määrittely tuoteolion avulla
- Käyttäjäoikeuksien hallinta projekteittain ja tuotteittain

- Offline-käytön mahdollisuus tai ohjelmistoversioiden kopiointi ulkoiselle tallennusvälineelle
- Varmuuskopioiden palauttaminen

Tavoitteiden ja vaatimusten määrittelyn pohjalta lähdettiin valitsemaan toteutukseen soveltuvia työkaluja.

### 4.3.2 Työkaluvaihtoehdot ja valitun työkalun esittely

Dokumenttien ja PC-ohjelmistojen versionhallintaan on olemassa paljon avoimen lähdekoodin työkaluja ja kaupallisia työkaluja. PLC-ohjelmistojen versionhallintaan erikoistuneita työkaluja on vain muutamia. Ne ovat kaupallisia ratkaisuja, joiden hankinta- ja ylläpitokustannukset ovat korkeita.

Tässä tutkimuksessa kartoitettiin yrityksessä jo käytössä olevien tietotyökalujen lisäksi avoimen lähdekoodin versionhallintatyökalujen, kuten Git ja Subversion ja kaupallisten PLC-ohjelmistoihin erikoistuneiden versionhallintatyökalujen soveltuvuutta automaatio-ohjelmistojen versionhallintaan. Yrityksessä on testattu käytössä olevan dokumentaationhallintatyökalu M-Filesin soveltuvuutta PLC-ohjelmistojen versionhallintaan. Testeissä osoittautui, ettei M-Files kykene täyttämään versionhallinnan tavoitteita ja vaatimuksia.

Avoimen lähdekoodin versionhallintatyökalut on tarkoitettu pääsääntöisesti vain tekstitiedostoista koostuvan lähdekoodin hallintaan. Siemensin PLC-alustalle tuotetut ohjelmistoprojektit ovat binääritiedostoja, joten avoimen lähdekoodin versionhallintatyökalujen ominaisuuksien ovat riittämättömiä PLC-ohjelmistojen hallintaan.

PLC-ohjelmistojen versionhallintaan tarkoitettujen työkalujen joukosta, löydettiin kaksi Siemensin PLC-alustojen kanssa yhteensopivaa työkalua: saksaisen AUVENSY-yhtiön versiondog ja yhdysvaltalaisen MDT Softwaren AutoSave. Molempien työkalujen suurimpia etuja olivat mahdollisuus vertailla graafisia PLC-ohjelmia ja PLC-koodiin tehtyjen muutosten automaattinen seuranta. Vertailemalla ohjelmointivirheet löytyvät nopeasti ja ohjelmistoversioiden väliset eroavaisuudet saadaan dokumentoitua. Automaattisella muutostenseurannalla voidaan havaita ohjelmistokoodiin tehdyt luvattomat muutokset sekä palauttaa ohjelmisto takaisin viimeisimpään testattuun ja stabiiliin versioon.

Muutostenseuranta-ominaisuus ei kuitenkaan kuulunut kummankaan työkalun perusversioon. Ominaisuuden käyttöönotto vaatisi työkalujen laajentamista maksullisilla lisäosilla ja erillisen palvelimen asentamista valvottaville laitoksille. Tällöin versionhallintaratkaisujen kustannukset nousisivat huomattavasti. Myös kaupalliset versionhallintaratkaisut suljettiin pois vaihtoehtojen joukosta, korkeiden hankinta- ja ylläpitokustannusten vuoksi.

Sulkemalla pois muut ratkaisut, yrityksessä jo käytössä oleva teknisen dokumentaation hallinnan PDM-työkalu Aton todettiin potentiaalisimmaksi ratkaisuksi ohjelmistojen versionhallintaan. Tällöin saavutettiin myös vaatimus ohjelmistojen versionhallinnan integroinnista osaksi tuotetiedon hallintaa.

Käytössä on siis vain yksi työkalu, jolla hallintaan kaikkea tuotantolinjaan liittyvää teknistä tietoa. Muutosten ja versioiden seuraaminen helpottuu. Lisäksi organisaation työntekijät ovat tottuneet käyttämään tätä työkalua osana päivittäistä toimintaa.

Roima Intelligencen toimittama Aton on suomalainen PDM-työkalu. Atonilla voidaan hallita tuotteeseen liittyviä nimikkeitä, nimikerakenteita, dokumentteja ja komponentteja. Atonissa dokumentit ja tuotekomponentit liitetään nimikkeisiin, joilla hallitaan tuotetietoa. Nimikkeistä voidaan muodostaa moduuleja ja liittää osaksi laajempaa tuoterakennetta. Tällöin tuotteesta luodaan oliopohjainen tuotekuvaus eli tuoteolio. Tuotteen ominaisuudet yksilöidään versiotunnusten ja sarjanumerorakenteiden avulla.

Nimikkeet mahdollistavat myös versioiden ja suositeltavuustietojen vertailun sekä elinkaariajattelun, joten Atonia voidaan käyttää tuotteen versioiden hallintaan ja tuoteversioiden kannattavuuden seurantaan läpi tuotteen koko elinkaaren.

Modulaarinen tuoterakenne mahdollistaa sekä massatuotettujen tuotteiden että räätälöityjen tuotteiden hallinnan. Tuotteet konfiguroidaan valitsemalla halutut moduulit tuoterakenteeseen. Aton sisältää työtiloja ja kansiorakenteita, joita voidaan hyödyntää tuoteperheiden ja tuotetarjoaman hallinnassa.

Työnkulkujen suunnittelu ja tilakirjanpito mahdollistavat vaiheittaisen muutostenhallintaprosessin sekä tuotteeseen kohdistuvien muutospyyntöjen ja asiakaspalautteiden hallinnan. Työkalulla voidaan seurata tietylle asiakkaalle toimitettuja tuotteita ja niihin tehtäviä muutoksia. Muutosten hallinnan lisäksi Atonissa voidaan perustaa ja hallita kantaversioita.

Nimikkeisiin liittyvät nimikekortit mahdollistavat tiedon merkityksen kuvailun attribuuteilla. Nimikekorttien avulla voidaan myös seurata, missä projekteissa tiettyä tuotetta tai sen komponentteja on käytetty. Aton siis mahdollistaa revisioiden hallinnan lisäksi myös varianttien hallinnan.

Hakutoiminto mahdollistaa tiedon nopean löytämisen sekä hakutulosten ryhmittelyn ja suodatuksen. Tiedot voidaan viedä Atonista ulkoiseen tallennusvälineeseen. Ulosviedyt tuotetiedot on mahdollista paketoida kokonaisiksi tuotteiksi tai tuotemoduuleiksi. Aton tukee siis myös simulaattorikehitystä.

Atonissa käyttäjäoikeuksia hallitaan työtilakohtaisesti. Tuotetieto voidaan jakaa kaikille tarvitsijoille, sijainnista riippumatta, käyttäjäoikeuksien hallinta –toiminnon ja jakeluver-

kosto-ominaisuuden avulla. Atonissa on mahdollista lukita tuotekomponentti tai dokumentti muokkauksen ajaksi. Lukitus voidaan vapauttaa muokkauksen päätyttyä. Näin ollen Atonia voidaan käyttää myös hajautettuun ohjelmistokehitykseen.

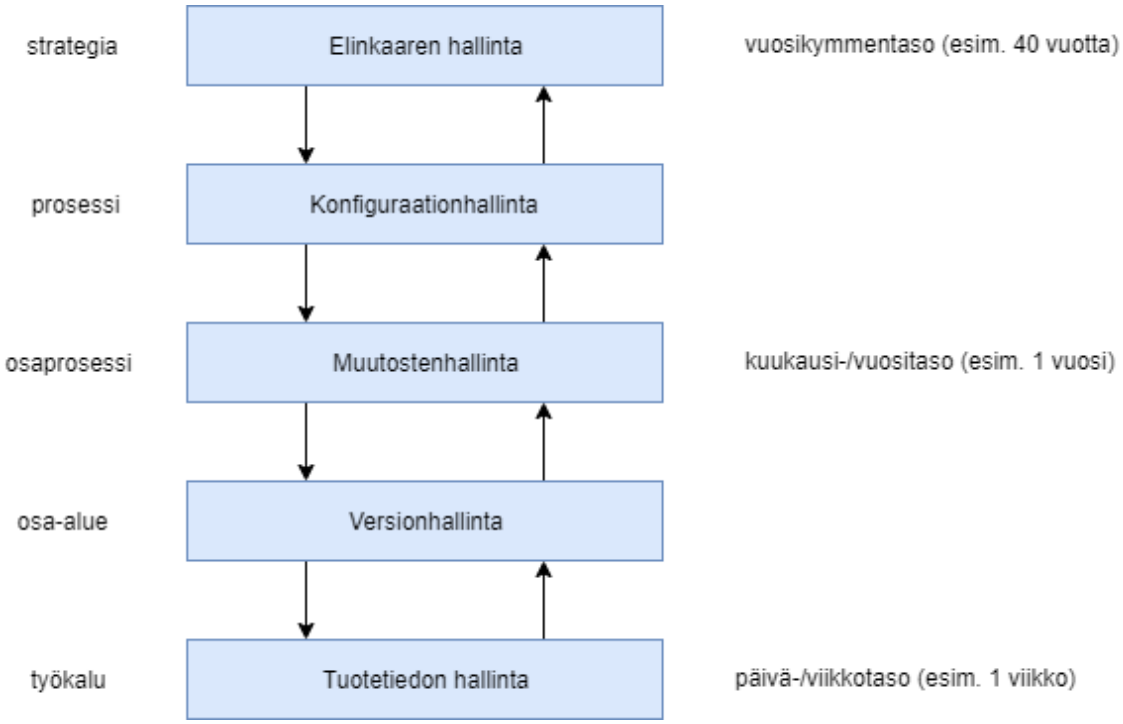
Atonin elinkaaren hallintaan liittyvät toiminnot mahdollistavat valmistuksen aikataulutuksen ja valmiusasteen seurannan sekä kustannusten allokoinnin. Työkalu soveltuu siis myös ohjelmointityön aikataulutukseen ja kustannusten hallintaan. Atonin keskitettyyn palvelimeen perustuva tiedon varastointi huolehtii tallennettujen tietojen varmuuskopioinnista ja alkuperäisen kopion suojaamisesta.

Atoniin on toteutettu standardirajapintoja moniin kaupallisiin suunnittelu- ja dokumentointityökaluihin. Esimerkkejä integroiduista työkaluista ovat toiminnanohjaustyökalut, CAD-työkalut ja Microsoft Officen toimistotyökalut. Atoniin voidaan implementoida itse tehtyjä ohjelmointirajapintoja. Aton soveltuu siis keskitetyksi tietokantaratkaisuksi, joka liittää yhteen kaikki käytössä olevat suunnittelutyökalut.

Edellä kuvattujen Atonin ominaisuuksien perusteella voidaan todeta, että se täyttää luvussa 4.3.1 versionhallintatyökalulle asetetut tavoitteet ja vaatimukset. Seuraavassa alaluvussa käsitellään modulaariselle tuotantolinjalle kehitettyjä versionhallintaratkaisuja. Ratkaisut sisältävät tuotantolinjan muutostenhallintaprosessin kuvauksen ja PLC-ohjelmistojen versionhallinnan menettelyt sekä tuotantolinjalle valitun tuoterakenteen kuvailun.

### 4.3.3 Versionhallinnan konsepti

Tuotantolinjan elinkaaren hallinta voidaan ajatella hierarkkisena hallintatehtävänä, joka kuvaa päätöksenteon hierarkiaa. Lähestymistavassa suunnitteluprosessit jaetaan hallintatasoihin hallintatehtävien aikahorisonttien mukaisesti: ylemmän tason hallintatehtävällä on pidempi suunnittelu ja toteutusaika verrattuna alempaan tasoon. Intuitiivinen hallintarakenne mahdollistaa kompleksisen tehtävän jakamisen yksinkertaisempiin osatehtäviin. Elinkaaren hierarkkinen hallintaprosessi on esitetty kuvassa 33.




***Kuva 33.*** *Elinkaaren hallinnan hierarkkinen rakenne.*

Kuvassa 33, päätöksenteon ohjaukset kulkevat ylhäältä alas. Päätöksenteon ohjauselementtejä ovat tehtävät, tavoitteet ja rajoitteet. Alemmilla tasoilla tehdyt päätökset eivät voi syrjäyttää ylemmillä tasoilla tehtyjä päätöksiä. Mittaukset kulkevat alhaalta ylös. Mittauselementit ovat tehtävien tilatietoja ja saavutettuja tuloksia. Ylempien tasojen ohjaus- ja mittaussignaalit ovat abstrakteja. Signaalit muuttuvat yksityiskohtaisemmiksi siirryttäessä matalammille hallintatasoille.

Elinkaaren hallinta on strategia ja ajattelutapa, joka määrittää systeemin versionhallinnan aikahorisontin. Konfiguraationhallinnan avulla moduuleista voidaan muodostaa kokonaisia systeemejä. Konfiguraationhallinnan tehtävänä on siis mahdollistaa versioiden testaaminen ja ylläpito. Ohjelmistojen näkökulmasta, konfiguraationhallinta helpottaa koontiversioiden muodostamista ohjelmistomoduuleista ja moduulien komponenteista. Muutostenhallinnan tehtävänä on varmistaa, että systeemiin tehdyt muutokset suoritetaan oikeaoppisesti. Laitteistoon kohdistuvien muutosten vaikutukset ohjelmistoon tulee siis analysoida. Analyysien perusteella päätetään muutosten hyväksymisestä tai hylkäämisestä. Muutostenhallinnalla myös seurataan, hyväksyttyjen muutosten, implementointityön etenemistä. PDM-työkalu on versionhallinnan toteutukseen käytetty ratkaisu. Tuotetiedon hallinnan tehtäviä ovat siis muun muassa versioiden merkitseminen ja tallentaminen sekä kantaversioiden hallinta.

Versiotunnukset ovat versionhallinnan tilatietoja, jotka kulkevat matalammalta abstraktiotasolta kohti korkeampaa abstraktiotasoa: versiot siirtyvät komponenteilta moduuleille ja edelleen konfiguroituihin tuotteisiin. Hierarkkisen tuoterakenteen alemmilla tasoilla

tehdyt versiomuutokset siis päivittyvät tuoterakenteen ylimmälle tasolle asti. Näin varmistetaan, että tuoteversiot pysyvät ajantasaisina ja muutokset ovat helposti PDM-työkalun käyttäjän havaittavissa.

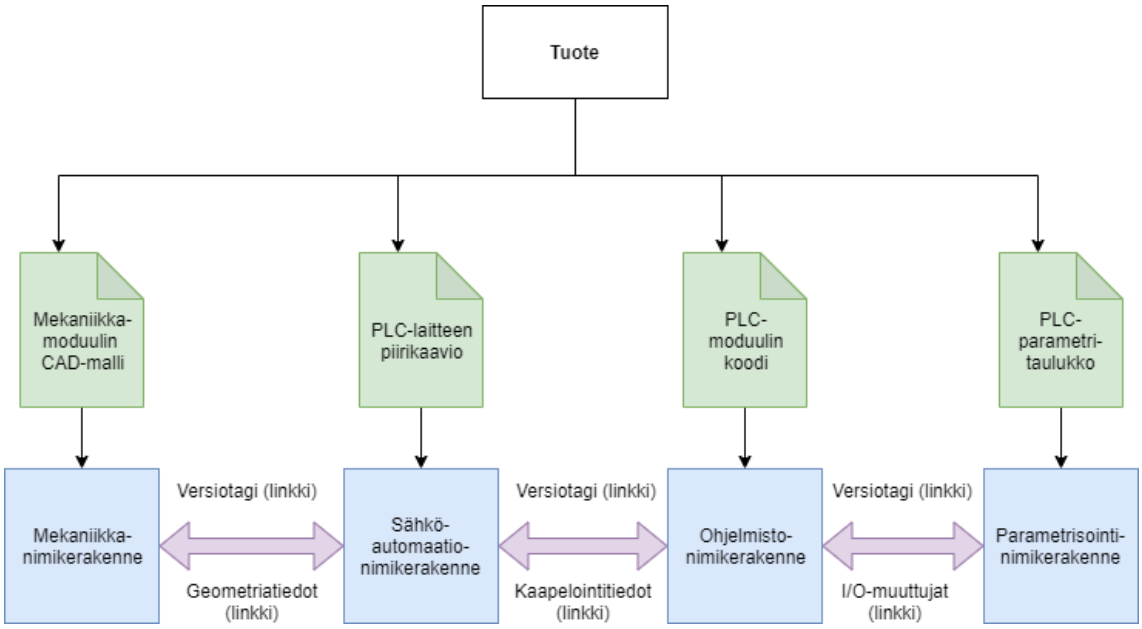
Versiotunnukset on valittu vastaamaan jo käytössä olevia mekaniikkasuunnittelun versiotunnuksia. Versiotunnus on kaksitasoinen, koostuen kirjaimista ja numeroista (esim. A.1 tai B.3). Käyttämällä samaa versiointimenettelyä kaikille tuoterakenteen tiedoille, voidaan yhdenmukaistaa ja selkeyttää muutosten ja versioiden hallintaa.

Ohjelmistojen versioinnissa käytetään tyypillisesti kolmitasoista numeerista versiotunnusta (esim. 4.9.6). Ensimmäinen taso (4) kuvaa taaksepäin yhteensopimatonta muutosta ohjelmistossa, kuten rajapinnan vaihtumista. Toinen taso (9) kuvaa uuden, taaksepäin yhteensopivan, toiminnon lisäämistä ohjelmistoon. Kolmas taso (6) taas kuvaa paikkausta, joka on taaksepäin yhteensopiva.

PDM-työkalun versiointikäytännöissä, nimikkeisiin liitettäville versioitaville tuotoksille annetaan yksilöllinen numeerinen tunniste eli oliotunniste (esim. 43510). Ensimmäistä julkaisua, jolle ei ole olemassa kantaversiota merkitään versiolla ”0”. Tämä julkaisu muodostaa kantaversion, jonka jälkeen seuraaviin julkaisuihin sovelletaan muutostenhallintamenettelyitä. Tällöin ensimmäistä muutosta kantaversioon merkitään versiolla ”A.1”. Kaksitasoista versiotunnusta käytetään kuvaamaan vain taaksepäin yhteensopivia muutoksia. Ohjelmistojen tapauksessa, kirjaintaso kuvaa uusia toimintoja ja numerotaso uusia paikkauksia.

Taaksepäin yhteensopimaton muutos, sen sijaan, katkaisee muutosketjun. Muutosketjun katketessa luodaan uusi olio, jonka versiointi alkaa alusta (versio 0). Vanha olio (oliotunniste 43510) linkitetään uuteen olioon (oliotunniste 43511) sopivalla versiotagilla (nimikekortin attribuutti), kuten ”kopioitu oliosta 43510”. Toisin sanoen, taaksepäin yhteensopimattomuutta tai epävarmuutta taaksepäin yhteensopivuudesta kuvataan PDM-työkalussa uudella oliotunnisteella.

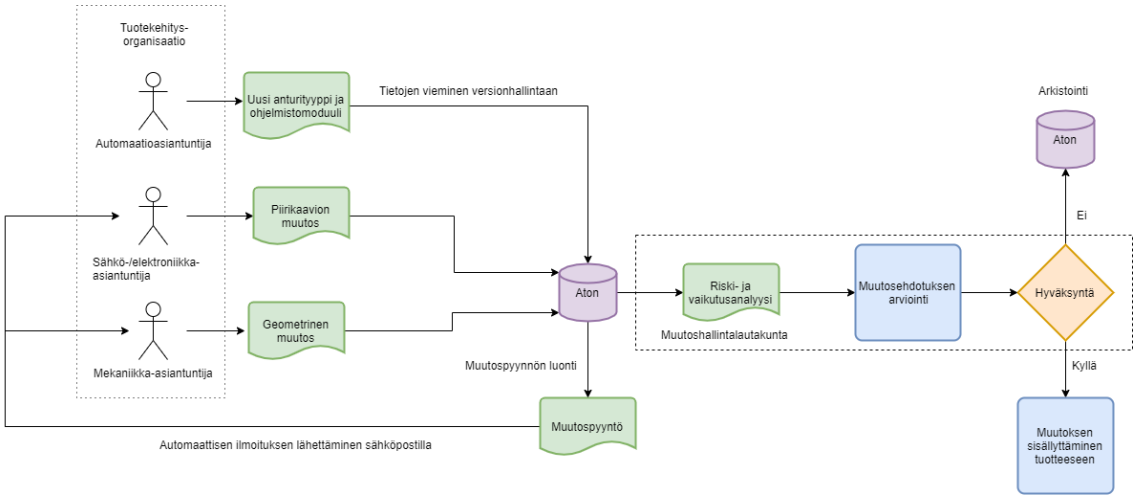
Tuoterakenteen jokainen moduuli versioidaan erikseen. Versioitujen moduulien välisiä linkkejä (versiotagi) seuraamalla voidaan jäljittää yhteensopimattomat rajapintamuutokset. Modulaarinen tuotantolinja koostuu useasta eri tuotteesta. Kukin tuote jakautuu edelleen mekaanisen prosessilaitteen moduuleihin, sähköautomaatiolaitteiden moduuleihin ja ohjelmistomoduuleihin sekä tuotteen parametreihin. Tuoterakenteen valinnassa on mukailtu kuvassa 19 esitettyä mekatronisen systeemin tuoterakennetta. Kirjallisuuden ratkaisumallia on jatkokehitetty huomioimalla myös tuotteeseen liittyvät parametrit sekä määrittelemällä nimikerakenteiden väliset rajapinnat. Valittu tuoterakenne on esitetty kuvassa 34. Kuvassa on myös havainnollistettu nimikerakenteiden nimikkeisiin liitettäviä dokumentteja.




***Kuva 34.*** *Modulaarisen tuotantolinjan tuoterakenne.*

Kuvasta 34 nähdään, että mekaniikka- ja sähköautomaatiokomponenttien väliset rajapinnat ovat CAD-dokumenttien sisältämiä geometriatietoja. Sähköautomaatiokomponenttien ja ohjelmistokomponenttien väliset rajapinnat ovat piirikaavioiden määrittelemiä kaapelointitietoja. Funktiolohkojen rajapintasignaalit (I/O-muuttujat) kuvaavat PLC-ohjelmiston toimintojen ja fyysisten komponenttien, eli mekaniikka- ja sähköautomaatiokomponenttien, välisiä riippuvuuksia. Toisin sanoen mekaanisen rakenteen kinematiikka sekä kenttälaitteiden ohjaus- ja mittaussignaalit vaikuttavat logiikkaohjaimella (PLC) ohjattavan systeemin dynaamiseen käyttäytymiseen. Jos yksikin edellä mainituista rajapinnoista muuttuu, muutosten vaikutukset pitää siis tarkistaa koko systeemin näkökulmasta. Muutosten vaikutuksia tarkastellaan määrämuotoisen muutostenhallintaprosessin avulla. Muutostenhallintaprosessi kuvattiin yleisellä tasolla luvussa 2.1.3. Konkreettisena esimerkkinä voidaan käyttää luvussa 4.2.2 esitettyä dynaamisen nopeuden säädön konseptia ja sen soveltamista poistokuljettimen energiankulutuksen optimointiin. Kuvassa 35 on havainnollistettu muutostenhallintamenettelyä, jossa tuotteeseen lisätään uusi automaatiotekninen muutospaketti. Muutospaketti sisältää laseranturin, joka mittaa poistokuljettimen materiaalikertymää. Ohjelmistomoduuli sisältää ominaisuuden, jolla estimoidaan poistokuljettimen materiaalikertymäprofiilia laseranturin mittaustiedon perusteella. Kun anturi lisätään, tuotteen kaapelointitietoja on muutettava. Uusi anturi aiheuttaa muutoksia myös CAD-mallissa, johon lisätään geometriatieto anturin asennuskohdasta. Lisäksi parametrisointitiedot muuttuvat, sillä anturin lisääminen vaatii uuden tulomuuttujan lisäämistä PLC-parametritaulukkoon. Anturin kalibrointi tai ohjelmistomoduulin sisäisten

muuttujien, kuten raja-arvoparametrien, muuttaminen eivät vaadi muutostenhallintamenettelyä, sillä nämä muutokset eivät vaikuta kuvan 34 mukaisiin rajapintoihin.




***Kuva 35.*** *Esimerkki muutostenhallintaprosessista*

Kuvasta 35 nähdään, että Atoniin tulee luoda työnkulku, joka mahdollistaa muutospyyntöjen luomisen, arvioinnin ja hyväksynnän. Integroidulla tuotekehitysprosessilla, voidaan parantaa tuotekehittäjien välistä yhteistyötä. Tällöin tuotteeseen kohdistuvasta muutoksesta informoidaan eri vastuualueiden asiantuntijoita, esimerkiksi sähköpostiviestin välityksellä. Sähköpostiviestin linkillä asiantuntijat pääsevät tarkastamaan tehtävälistalta omat muutokseen liittyvät vastuutehtävänsä.

Atoniin vietyjen muutosdokumenttien perusteella voidaan määritellä muutoksen implementointikelpoisuus. Hylätty muutospäätös ja siihen liittyvä analyysi arkistoidaan. Näin voidaan kehittää tarvittavat korjauspiteet. Mikäli muutoslautakunta hyväksyy muutokset, ne sisällytetään tuotteeseen.

Systeemivastaavan tulee osallistua kriittisiin muutospäätöksiin. Tuotekehitysorganisaatiosta valittu systeemivastaava toimii muutoshallintalautakunnassa. Systeemivastaava voi toimia myös konfiguraationhallintavastaavana. Ideaalitapauksessa muutostenhallintaan osallistuu henkilöstöä useilta eri osastoilta: esimerkiksi tuotekehityksestä (tuotekehitysinsinöörit), projektitoiminnasta (projektipäällikkö ja eri alojen pääinsinöörit) sekä huolto- /palveluliiketoimintaosastolta ja talousosastolta. Muutosten vaikutuksia arvioidaan siis eri alojen asiantuntijoiden (esim. sähkö-, automaatio- ja mekaniikkainsinööri sekä projektipäällikkö) kesken. Näin varmistetaan erilaisten näkökulmien huomiointi ja mahdolliset eturistiriidat toimijoiden välillä. Tällöin voidaan minimoida muutosten vaikutukset ja muutostarpeet systeemin muihin komponentteihin.

Yleensä muutosehdotus tulee tuotteen parissa toimivalta henkilöltä. Muutosehdotus voi tulla myös asiakkaalta tai muun sidosryhmän edustajalta. Muutostenhallintamenettelyn ensisijainen hyöty on muutosehdotusten systemaattinen hallinta. Tällöin ehdotettuja muutoksia arvioidaan koko systeemin kannalta, eikä vain yksittäisen toiminnallisuuden tai komponentin osalta. Muutostenhallinnalla varmistetaan myös muutosten oikeanlainen implementointi, aikataulu- ja kustannusrajoitteet huomioiden. Muutostenhallinnassa hyväksytyt tuotemuutokset julkaistaan versionhallintatyökalulla. Modulaarinen ohjelmisto vaatii siis myös julkaisumenettelyn.

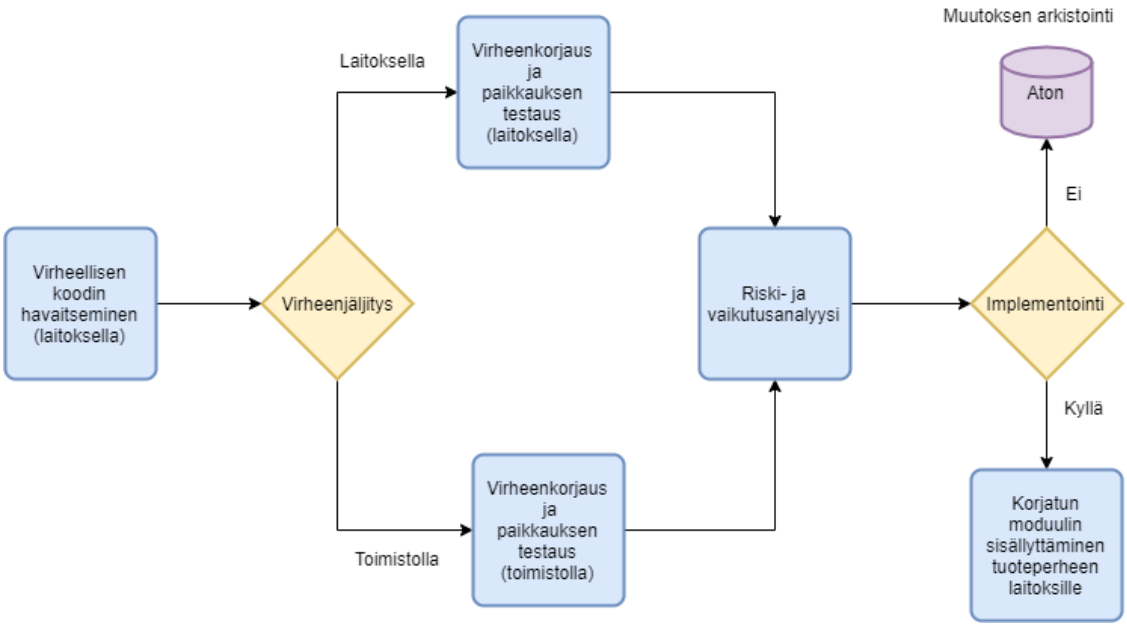
Kuvan 30 ohjelmistomoduuli voi sisältää esimerkiksi Kalman-suotimen implementoinnin, jonka viritykseen käytetään optimointialgoritmia. Yksinkertaistettuna, Kalman-suotimen tehtävänä on suodattaa staattista mittauskohinaa mittaustuloksen tarkkuuden parantamiseksi. Kalman-suotimen suunnitteluun ja viritykseen voidaan käyttää Mathworksin MATLAB- ja Simulink-työkaluja. Tällöin uusi ominaisuus voidaan validoida simuloimalla, ennen laitokselle vientiä. Käyttöönotossa ja SAT:ssa tulee paikata moduuli- ja integrointitestauksessa sekä FAT:ssa piiloon jääneet ohjelmistovirheet. Paikkaukseen liittyy aina jonkinasteinen riski, riippumatta siitä toteutetaanko korjaus vai ei. Riskiä voidaan madaltaa systemaattisella testauksella, jossa hyödynnetään testikattavuusanalyysejä.

Jotkin ohjelmistovirheet havaitaan vasta, laitoksella, käyttöönottovaiheessa. Tämä johtuu simulointimallin epätäydellisyydestä. Simulointimalli ei siis huomioi kaikkia fysikaalisia rajoitteita ja ilmiöitä. Simulointitestauksella (kuten HIL ja FAT) ja fyysisellä testauksella (SAT) on omat erityispiirteensä. Simulointitestien kattavuutta rajoittaa mallinnukseen ja testiskenaarioihin käytettävissä oleva aika. Laitoksella tapahtuvan ohjelmistokehityksen ja testauksen kustannukset ovat huomattavasti toimistolla tapahtuvaa testausta korkeammat. Ohjelmistojen paikkaamista laitoksella rajoittaa siis kustannus- ja aikataulupaineet. Myös asiakastyytyväisyys kärsii pitkittyneestä käyttöönotosta. Asiakkaan luona kehitetty koodi on myös tyypillisesti huonolaatuista. Paineen alaisena, ohjelmistokehittäjän, on vaikea noudattaa hyviä ohjelmointitapoja ja sovittuja ohjelmointikäytäntöjä. Lisäksi haasteena on laitoksella tapahtuvien muutosten dokumentointi ja niiden saattaminen (toimistolla sijaitsevaan) versionhallintaan. Tällöin riskinä on, että toimistolla ja laitoksella on eri ohjelmistoversiot.

Kuvassa 36 on havainnollistettu, miten simulointia voidaan hyödyntää myös laitoksella SAT-testauksessa. Mikäli laitoksella havaitaan ohjelmistovirhe, virheen paikantamiseen ja paikkaukseen voidaan pyytää avuksi toimistolla olevaa tuotekehitysinsinööriä. Yhteys laitoksella olevan käyttöönottoinsinöörin ja toimistolla olevan tuotekehitysinsinöörin välille voidaan järjestää esimerkiksi etä-, video- tai puhelinyhteydellä. Simulointia hyödyntävä tuotekehitysinsinööri on rennommassa ilmapiirissä ja näin ollen pienemmän paineen alaisena. Lisäksi simulaattorilla voi ajaa erilaisia testiskenaarioita huomattavasti

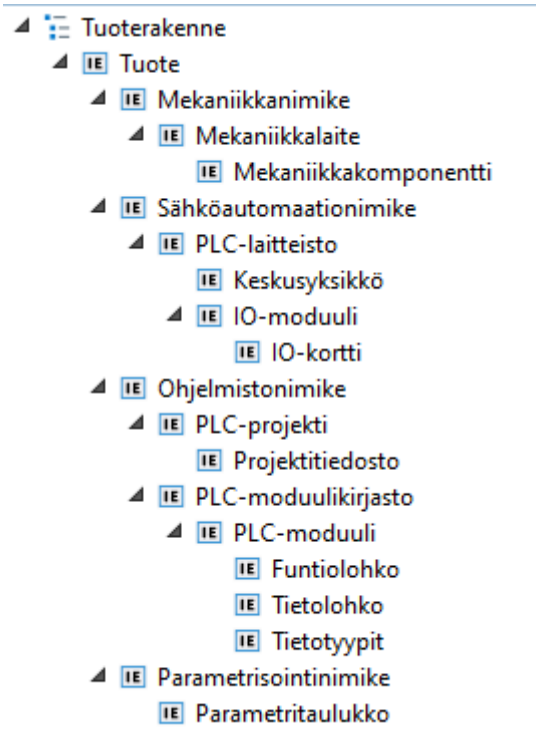
helpommin ja nopeammin kuin fyysisellä laitteistolla. Simulointia hyödyntämällä, käyttöönottoa voidaan nopeuttaa ja samalla parantaa koodin laatua ja ylläpidettävyyttä. Reaaliaikainen yhteys toimistolle varmistaa muutostietojen siirtymisen laitokselta versionhallintaan.

Mikäli tehty paikkaus aiotaan implementoida myös muille jo käytössä oleville laitoksille, tulee ohjelmistopaikkaukseen liittyvä riski ja paikkauksen kriittisyys määrittää. Muutoksen implementointi voi olla perusteltua, jos ongelma ilmenee muillakin laitoksilla. Paikkauksella voidaan saavuttaa myös suorituskykyparannuksia. Tällöin paikkauksen voi luokitella (kriittisyystasosta riippuen) myös myytäväksi ohjelmistopäivitykseksi.



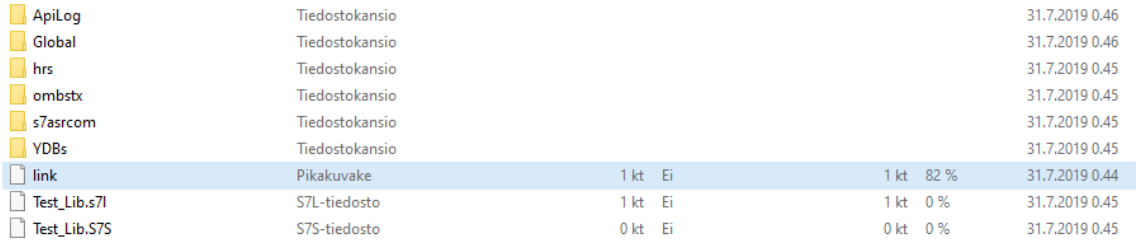

***Kuva 36.*** *Ohjelmistomoduulin julkaisumenettely.*

Kehitettyjen moduulien uudelleenkäyttö vaatii soveltuvia moduulikirjastoja. PLC-moduulit koostuvat ohjelmistokomponenteista. Kuten luvussa 4.2.4 todettiin, PLC-moduulin ohjelmistokomponentteja ovat funktiolohko, tietolohko ja käyttäjän määrittelemät tietotyypit. Kuvassa 37 on esitetty kuvan 34 mukainen tuoterakenne yksityiskohtaisesti. Rakenne koostuu laitteiston ja parametrien lisäksi PLC-ohjelmiston kantaversion mukaisesta projektitiedostosta ja moduulikirjastosta. Tuoterakenteen määrittelyyn voidaan käyttää, kuvan 2 mukaista, konfiguraation identifiointiprosessia. Kantaversioita voidaan perustaa, ohjelmistokehitysprojektin edetessä, kuvan 4 mukaisesti. Huomionarvoista kuitenkin on, että erilaisia kantaversioita voitaisiin perustaa enemmänkin. Tarvittavien kantaversioiden määrä ja sisältö riippuvat käytetyistä ohjelmistokehityksen menetelmistä.




***Kuva 37.*** *Tuoterakenne puuhierarkiana.*

TIA Portal (Step 7) –työkalulla I/O-muuttujista voidaan muodostaa parametritaulukoita Export-toiminnolla. Tällöin I/O-muuttujat tallennetaan Excel-taulukoksi. Excel-taulukon rakenne on taulukon 7 mukainen. PLC-moduulien tai niiden komponenttien lähdekoodit voidaan tuoda tekstitiedostoina versionhallintaan. TIA- Portal –työkalussa on ”Generate source from blocks”-toiminto. Toiminto muuntaa funktiolohkon ja tietolohkon toteutukset structured text –ohjelmointikielen mukaisiksi lähdekooditiedostoiksi. Structured text perustuu Pascal-ohjelmointikieleen. Lähdekoodien tiedostonimet ovat vapaasti määritettävissä. Tietotyypit voidaan kopioida TIA Portalin käyttöliittymässä kopioi-toiminnolla ja liittää tekstitiedostoon. Tekstitiedosto kannattaa tallentaa CSV-muodossa (engl. Comma Separated Values, CSV), jolloin sen voi avata helposti Excel-työkalulla.
Vaihtoehtoisesti komponenteista ja moduuleista voidaan luoda Step 7-kirjastoja. Tällöin kirjastot on pakattu binääritiedostoiksi kansiorakenteeseen. Tämän lähestymistavan heikkoutena on, että kirjastojen sisällön voi nähdä vain suunnittelutyökalun (TIA Portal) käyttöliittymässä, eikä kirjastojen sisältöä voi tarkastella versionhallintatyökalulla. Kuvassa 38 on havainnollistettuna TIA Portalilla luodun Step 7 –kirjaston kansiorakenne.




***Kuva 38.*** *Esimerkki Step 7 –kirjastosta.*

Myös Step 7 –ohjelmistoprojektit tallennetaan binääritiedostoina kansiorakenteisiin. Toisin kuin ohjelmistokomponenttien tapauksessa, ohjelmistoprojekteista ei voi generoida lähdekoodia. Projektitiedostot sisältävät PLC-alustalle luodun kohdekoodin eli ohjausso-velluksen ja projektiin liittyvän PLC-laitteistokonfiguraation määrittelyn. Kuvassa 39 on esitetty Step 7 –ohjelmistoprojektin kansionrakenne.


***Kuva 39.*** *Esimerkki Step 7-ohjelmistoprojektista.*

Kuten luvussa 4.2.3 todettiin, laitteistokonfiguraation määrittelyn voi tallentaa myös AutomationML-tiedostona. Tällöin laitteistokonfiguraatiotiedosto voidaan avata TIA-portal –työkalun lisäksi Eplan-työkalulla tai tekstinkäsittelytyökalulla (kuten Microsoft Word). AutomationML-laitteistokonfiguraatiotiedosto on siis tarkasteltavissa myös Aton-versionhallintatyökalulla.

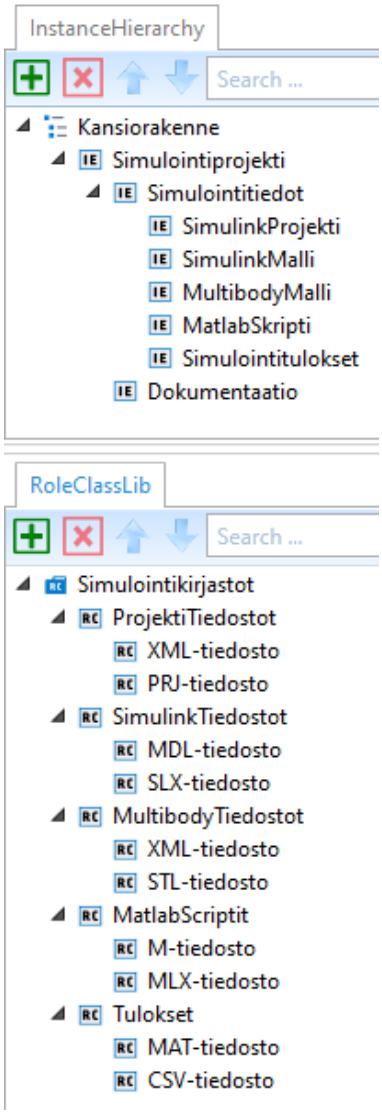
Projektitiedostot sisältävät suuren määrän binääritiedostoja, joiden merkitystä ja sisältöä on vaikea tulkita TIA Portal –työkalun käyttöliittymän ulkopuolella. Olemassa olevien, ei-modulaaristen, ohjelmistojen kohdekoodi kannattaa siis viedä versionhallintaan yhtenä ZIP-pakkauksena. Tällöin ohjelmistoprojekti voidaan tuoda versionhallinnasta helposti suunnittelutyökalulle. Binäärimuotoisten kohdekooditiedostojen vuoksi, ohjelmistoversioiden välisiä eroavaisuuksia voi tarkastella ainoastaan TIA Portalin käyttöliittymässä.

Ohjelmistoprojektin sisältävään ZIP-pakkaustiedostoon on viitattu kuvassa 37 termillä projektitiedosto.

ZIP-pakkausten käyttämisen heikkoutena on suuri tallennustilan tarve, mikäli ohjelmistoja muokataan usein. Uusia ominaisuuksia lisätessä, syntyy redundanttista tietoa. Tämä johtuu siitä, että vanhat ominaisuudet ovat uuden pakkauksen lisäksi myös edellisten versioiden pakkauksissa.

Myös ohjelmistojen kehitykseen käytettävät simulointimallit ja simulaatioihin liittyvät muut tiedostot tulee viedä versionhallintaan. Simulointiprojektit voidaan tuoda Simulink-työkalulta Simulink-projektikansioina. Simulink-työkalun projektitoiminto on ominaisuus, jolla mallipohjaisen suunnittelun kehitysalustaa voidaan yhdenmukaistaa. Yhdenmukaisen alustan ansiosta simulointimallit ja testiskenaariot ovat helpommin käytettävissä. Projekteihin voidaan tallentaa räätälöityjä kirjastoja ja kansiorakenteita. Projekti mahdollistaa myös mallin konfiguroinnin automatisoinnin. Malliin liittyvät konfiguraatiotiedostot voidaan ladata automaattisesti simulointiympäristöön, kun projekti viedään versionhallinnasta Simulink-työkalulle. Projektit mahdollistavat myös versionhallintatyökalun integroinnin simulointiympäristöön. Simulink-projektit tarjoavat standardirajapinnat Git- ja Subversion-työkalulle. Simulink-työkalu voidaan integroida myös muihin versionhallintatyökaluihin implementoimalla räätälöity ohjelmointirajapinta.

Kuvassa 40 on esitetty ehdotus simulointiprojektin räätälöitävästä kansiorakenteesta ja siihen liitettävistä kirjastoista. 3D-simulointiprojekti sisältää Simulink-lohkokaaviomallin, CAD-malleista generoidut Simscape Multibody 3D-mallit sekä Simscape kirjastojen toimilaitteita ja antureita. Lisäksi projekti voi sisältää konfigurointitiedostoja MATLAB-skripteinä. Konfiguraatiotiedostot voivat olla esimerkiksi testiskriptejä tai parametrien määrittelyjä. Tulokset kannattaa tallentaa sekä MATLABin matriisitiedostoina (MAT-tiedosto) että tekstitiedostoina (esim. CSV-tiedosto). Tällöin simulointitulokset ovat myös muiden työkalujen (kuten Excel) käytettävissä. Myös parametrit kannattaa viedä Excel-taulukoina MATLAB-ympäristöön. Soveltuvalla MATLAB-skriptillä voi siis alustaa Simulink-mallin versionhallinnasta tuoduilla Excel-taulukoilla.

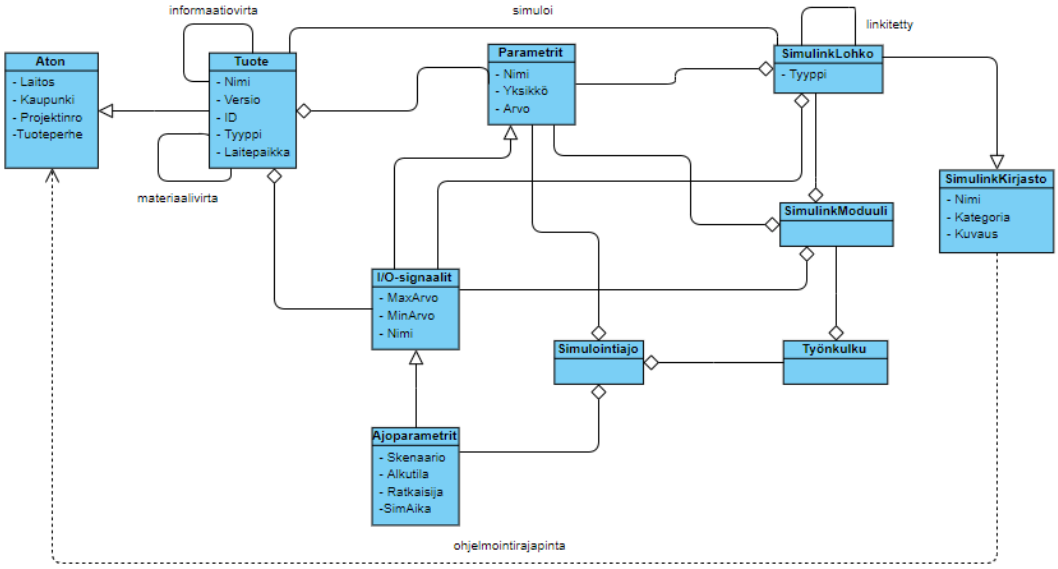



***Kuva 40.*** *Simulink-projektin rakenne.*

Versionhallinnassa kuvan 40 simulointitiedostot liitetään kuvan 34 tuoterakenteeseen. Simulointimalleja käytetään pääosin ohjelmistokehitykseen, joten projektikansiot kannattaa liittää ohjelmistonimikkeisiin. Parametritaulukot liitetään parametrisointinimikkeisiin. Näin voidaan erotella parametrien ja mallien versiot toisistaan. Tämä lisää mallien uudelleenkäytettävyyttä.

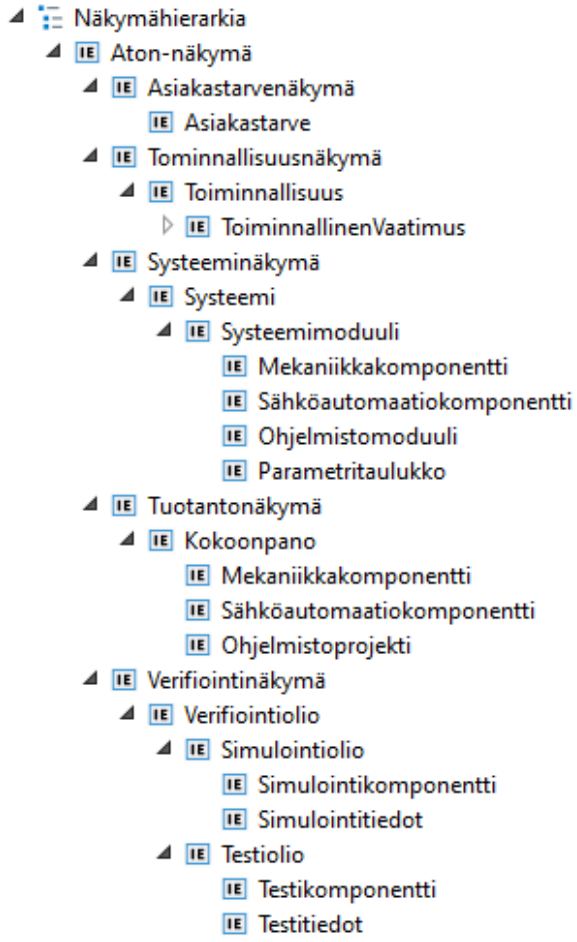
Versionhallintaan luoduista standardoiduista tuotteista ja tuoteperheistä voidaan muodostaa simulointimalleja. Standardoiduille laitteille kehitettävän simulaattorin uudelleenkäytettävyysaste on korkeampi verrattuna massaräätälöityihin laitteisiin. Kuvassa 41 on havainnollistettu (UML-luokkakaaviona), miten tuotetiedot ja simulointimallit liittyvät toisiinsa. Tuoteperheistä periytetyillä modulaarisilla tuotteilla on standardoidut rajapinnat. Simulink-lohkoista koostuviin simulointimoduuleihin tarvitsee syöttää vain IO-signaalien tunnistetiedot sekä simulointimallin karakterisoivat simulointiparametrit. Kun tuotteesta

luodaan simulointimalli, ensimmäistä kertaa, simulointilohkot ja moduulit tallennetaan Simulink-työkalulla luotaviin kirjastoihin. Nämä kirjastot voidaan viedä Simulink-projektina versionhallintaan. Kahdensuuntaisella tiedonvaihdolla, tuoteversioiden simulointimallit voidaan konfiguroida nopeasti versionhallintaan tallennettujen simulointikirjastojen ja parametritaulukoiden avulla.




***Kuva 41.*** *Tuotetietojen ja simulointimallien välinen yhteys.*

Mallipohjaista tuotekehitystä voidaan tukea sopivien näkymien avulla. Myös Aton mahdollistaa tällaisten näkymien luomisen. Kuvassa 42 on esitelty Atoniin perustettavia näkymiä. Näkymät on valittu kuvan 20 ratkaisumallin mukaisesti. Systeeminäkymään on kuitenkin lisätty malliin liittyvät parametrit. Poistokuljetin-tuotteen tapauksessa, systeeminäkymä vastaa kuvan 28 yksityiskohtaista mallia.




***Kuva 42.*** *Atonin näkymiä.*

Kuvan 42 testitiedot-luokkaan voidaan tallentaa mittaustietoa esimerkiksi FAT- ja SAT-testeistä sekä määräaikaistesteistä. Mittaustietoja voidaan tällöin käyttää simulointimallien parametrien identifiointiin. Näin simulointitulosten tarkkuus ja täsmällisyys paranevat. Kerätyn mittaustiedon perusteella on mahdollista luoda rekonstruktioita reaalimaailman tuotantolinjojen ja tuotteiden ongelmatilanteita. Mittaustietoa voidaan kerätä myös käyttövaiheesta, jolloin laitosten suorituskykyä voidaan vertailla. Vertailuanalyysien (engl. benchmarking) avulla voidaan kehittää ohjelmistoja mallipohjaisesti ja mittaustietoihin perustuen. Teolliseen tietoverkkoon (engl. industrial internet) pohjautuvat ratkaisut mahdollistavat mittaustiedonkeruun laitoksilta.

Liitteessä D on havainnollistettu kehitetyn versionhallintakonseptin toteuttamiskelpoisuutta. Atoniin on luotu kuvan 37 mukainen tuoterakenne murskaimelle. Murskaimeen liittyvä ohjelmistoprojekti on liitetty projektitiedosto-nimikkeeseen yhtenä ZIP-pakettina. Projektitiedosto-nimikkeeseen on myös liitetty ohjelmistoprojektin määrittelydokumentti Excel-taulukkona. Versionhallintakonseptin testaus ja implementointi eivät sisälly tähän tutkimukseen. Versionhallintamenettelyn käyttökelpoisuuden todentaminen vaatii vielä lisää tutkimusta.

## 4.4 Toimenpide-ehdotus

Luvussa esitetään toimenpiteitä, jotka edistävät ohjelmistokehitystä tutkitussa yrityksessä. Ensin käsitellään siirtymäprosessin strategian luomiseen liittyviä vaiheita, joilla ei-modulaarisista automaatio-ohjelmistoista voidaan siirtyä kohti modulaarisia ohjelmistoja. Tämän jälkeen, arvioidaan siirtymästä aiheutuvia kustannuksia ja siirtymäprosessin vaatimien investointien kannattavuutta. Lopuksi esitellään investointilaskelmiin liittyviä epävarmuustekijöitä ja arvioidaan estimaattien luotettavuutta.

### 4.4.1 Strategia modulaarisiin ohjelmistoihin siirtymiseksi

Tässä luvussa käsitellään siirtymäprosessia nykytilanteesta kohti modulaarisia ohjelmistotuoteperheitä. Modulaarisuus nopeuttaa ohjelmistokehitystä ja laskee näin ohjelmiston kustannuksia, tiettyyn rajaan asti. Moduulien määrän kasvaessa, myös integrointikustannukset kasvavat. Ohjelmistomoduulien määrä ja niiden käyttötarkoitukset ton siis määritettävä tarkkaan. Moduulien rakentamiseen liittyy monia rajoitteita, kuten mekaniikka- ja sähköautomaatiolaitteiden projektikohtaiset erot sekä turva-automaation vaatimukset. Erityisen haastavaksi modulaarisuuskonseptin soveltamisen tekee voimalaitostoimitusten projektiluontoisuus. Myös uusien menetelmien, taitojen ja työkalujen oppiminen vie paljon aikaa. Kustannustehokas siirtymä vaatii siis vahvaa sovellusosaamista ja tuotteen ominaisuuksien tuntemista sekä nykyaikaisten ohjelmistokehitysmenetelmien haltuunottoa.

Eräs lähestymistapa siirtymästrategian luomiseen on työpajojen järjestäminen organisaation ulkopuolisten asiantuntijoiden kanssa. Työpajojen avulla saadaan käytännön tietoa modulaariseen ja mallipohjaiseen suunnitteluun liittyvistä haasteista ja rajoitteista sekä hyväksi havaituista toimintatavoista. Näin voidaan asettaa realistiset tavoitteet ja vaatimukset siirtymäprosessille. Työpajatoiminnan avulla voidaan myös verrata ja analysoida sovellusten kehityskustannuksia. Näin voidaan kehittää perusta siirtymän kustannusten ja saavutettavissa olevien tuottojen arviointiin.

Voimalaitos- ja energiateollisuudessa ei tyypillisesti kehitetä modulaarisia ohjelmistoja, joten siirtymäprosessiin liittyviä kokemuksia ja suosituksia tulisi kartoittaa muilta toimialoilta. Modulaarisia PLC-ohjelmistoja käytetään esimerkiksi robotiikan liikkeenohjaussovelluksissa. Mallipohjaista suunnittelua käytetään tyypillisesti, mikrokontrolleripohjaisissa, sulautetuissa järjestelmissä. Mallipohjaisen suunnittelun kehittyneimpiä sovellusalueita ovat liikkuvien työkoneiden moottorinohjausjärjestelmät, laivojen ja lentokoneiden autopilottijärjestelmät sekä tieliikenneajoneuvojen lukkiutumattomat jarrutusjärjestelmät.

Työpajoihin voisi siis kutsua paikalle konsultteja, jotka ovat työskennelleet edellä mainittujen sovellusalueiden parissa.

Modulaarisen ohjelmiston kehittämistä ei tarvitse aloittaa tyhjästä. Vanhoja ohjelmistoja voidaan hyödyntää tuoteperhesuunnittelun pohjana. Analysoimalla olemassa olevien PLC-ohjelmistojen variantteja, saadaan yleiskuva tuotteen toiminnallisuuksista eli ominaisuusmalli. Tuotevarianteista voidaan luoda tuoteperheitä, varianttien yhteisiin ominaisuuksiin perustuen. Tuoteperhesuunnittelun tavoitteena on rakentaa tuoteportfolio, jonka tuotteet voidaan kehittää käyttäen samankaltaisia moduuleja ja menetelmiä.

Moduulien korkean uudelleenkäytettävyysasteen saavuttaminen edellyttää tuoteperheen varianttien systemaattista hallintaa. Ohjelmistojen välisiä eroavaisuuksia voidaan analysoida esimerkiksi lähdekoodien ja kuvan 32 mukaisten dokumenttien perusteella. Tarkasteltavaksi kannattaa ottaa muutamia hyvin toteutettuja projekteja. Analyysin tavoitteena on luoda käsitys laitteisto- ja ohjelmistoalustojen standardointipotentiaalista ja tuoteominaisuuksiin perustuvista varioitavuuden hallintamekanismeista. Varioitavuuden hallintaa edistää selkeästi määritelty systeemi- ja ohjelmistoarkkitehtuuri sekä muutostenhallintamenettely. Esimerkki muutostenhallintamenettelystä esiteltiin luvussa 4.3.3.

Arkkitehtuuri kuvaa korkean tason suunnittelua, jossa määritellään moduulit ja niiden rajapinnat. Arkkitehtuurin suunnittelussa tehdään valintoja, jotka tukevat ylläpidettävyyttä ja laajennettavuutta. Näitä valintoja ovat esimerkiksi moduulien laajuudet ja käyttötarkoitukset sekä moduulien välisten rajapintojen sopiminen. Tämän tutkimuksen luvussa 4.2 kuvattiin tällaisen arkkitehtuurin suunnittelua systeemi- ja laitteistotasolla. Läpileikkauksessa ei kuitenkaan määritelty ohjelmistoarkkitehtuuria, joten työpajoissa tulisi olla paikalla ainakin ohjelmistoarkkitehti ja mallipohjaisen suunnittelun asiantuntija.

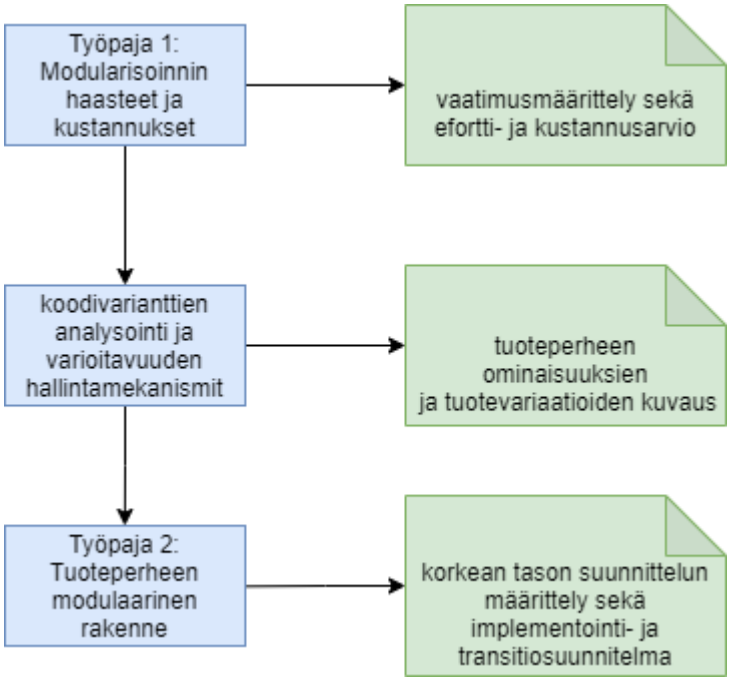
Olemassa olevia tai kehityksen alla olevia ohjelmistoprojekteja voidaan hyödyntää myös modulaarisen ohjelmiston implementointityössä. Vanhojen koodipohjien hyödyntämisessä voidaan käyttää refaktoroinnin menetelmiä. Refaktoroinnissa lähdekoodin toiminnallisuudet säilytetään, mutta ohjelmiston sisäistä toteutusta muutetaan jakamalla aiemmin toteutetut metodit pienempiin osiin. Näin voidaan muodostaa ohjelmistokomponentteja ja komponenteista koostuvia moduuleja. Refaktoroinnin menetelmiä ovat esimerkiksi olio-ohjelmoinnin mukaisten rakenteiden luominen (funktiolohkot, tietolohkot ja tietotyypit) sekä muuttujien eli parametrien uudelleen nimeäminen. Refaktoroinnissa voidaan hyödyntää luvussa 4.2 esiteltyjä ratkaisumalleja.

Ohjelmistoalustoille on saatavissa myös valmiita, avoimen lähdekoodin, komponentti- ja moduulikirjastoja. Yhdysvaltalainen konsultointiyritys DMC ylläpitää Siemensin TIA Portal –ohjelmointiympäristölle kehitettyjä avoimen lähdekoodin kirjastoa. Kirjaston nimi on Siemens Open Library. Kirjasto sisältää huolellisesti dokumentoituja funktiolohkoja (mo-

duulit) ja tietotyyppejä (rajapinnat) prosessiteollisuuden ja liikkeenohjauksen sovelluksiin. Kirjastossa on esimerkiksi moottorien ja taajuusmuuttajien sekä venttiilien ohjaukseen tarkoitetut toteutukset. Prosessien hallintaan on käytettävissä muun muassa PID-säätimen toteutus ja pinnankorkeuden monitorointitoiminto. Kirjastosta löytyy lukitusten toteutukseen sekä I/O-signaalien ja hälytysten hallintaan käytettäviä moduuleja ja rajapintoja.

Siemens Open Library tarjoaa esitestattuja moduuleja ja rajapintatoteutuksia, joiden lähdekoodia saa muokata, jakaa ja myydä vapaasti. Kirjasto voidaan määritellä kantaversioksi, jota muokkaamalla on mahdollista kehittää omia räätälöityjä kirjastoja. Siemens Open Library toimii siis standardoituna pohjaratkaisuna, jolla voidaan nopeuttaa siirtymää kohti modulaarisia ohjelmistoja.

Vanhojen koodipohjien ja avoimen lähdekoodin kirjastojen kartoittamisen jälkeen, voidaan laatia ohjelmistoarkkitehtuurin kuvaus sekä määritellä moduulien rakennuksen ja konfiguroinnin tekniset keinot. Modulaarisen ohjelmiston konfigurointikonsepti esiteltiin luvussa 4.2.4. Työpajojen ja koodianalyysien tulosten perusteella voidaan laatia suunnitelma siirtymän toteuttamiseksi ja modulaaristen tuoteperheiden implementoimiseksi. Edellä kuvatun siirtymästrategian luomiseen liittyvä työnkulku on esitelty kuvassa 43.




***Kuva 43.*** *Siirtymästrategian luominen.*

Tutkittavan yrityksen tapauksessa, kuvan 43 mukaisen, lähestymistavan toteuttamista vaikeuttaa resurssien rajallisuus. Ohjelmistot on tähän asti teetetty pääsääntöisesti alihankintana, jolloin ohjelmistokehityksen tietotaito ei ole karttunut organisaation sisällä.

Rajallisten resurssien takia, myös projektidokumentaation laatu vaihtelee. Erityisesti toiminnankuvaukset on laadittu liian yleisellä tasolla. Toiminnankuvauksia tulisi siis täsmentää yksityiskohtaisilla tuotteiden ja toimintojen kuvauksilla.

Eräs hyväksi havaittu lähestymistapa automaatiotoimintojen dokumentointiin on käyttää nelitasoista kuvausta. Ensin määritellään tuotteen toiminnalliset vaatimukset, jonka jälkeen luodaan yleiskuva toiminnallisuuksista. Kolmannella tasolla toiminnot kuvataan yksityiskohtaisesti. Viimeisenä tasona on implementaation kuvaus. Implementaatiokuvauksessa ohjauslogiikka kuvaillaan lohkokaavioina ja käytetyt parametrit taulukoina. Lähdekoodissa tulee käyttää hyviä nimeämis- ja kommentointitapoja. Taulukossa 7 havainnollistettiin systemaattista I/O-muuttujien nimeämis- ja kommentointitapaa.

Organisaation ulkopuolisten ohjelmistokehittäjien valvonta ja ohjeistus on myös ollut puutteellista, joten ohjelmistojen toiminnallisuuksien toteutustavat vaihtelevat projektista toiseen. Ohjelmistokehityksen pohjana on käytetty edellisten projektien toteutuksia. Kantaversioiden puutteen takia ohjelmistot on tuotettu kopioimalla ja muokkaamalla soveltuvia osia vanhoista toteutuksista, jonka seurauksena lähdekoodi on vaikeasti ymmärrettävää. Ohjelmistoissa ei myöskään ole käytetty selkeää rakennetta tai hyvien ohjelmointikäytäntöjen mukaista ohjelmointityyliä.

Edellä mainittujen syiden vuoksi, ohjelmistojen analysointi ja refaktorointi on haasteellista. Analysointivaiheen referenssiprojekteina tulisi siis käyttää tuleviin projekteihin kehitettäviä ohjelmistoja. Toinen työpaja kannattaakin järjestää vasta, kun ohjelmistokehitystä tukeva dokumentaatio ja kehitettävät ohjelmistot ovat paremmin laadittuja. Ohjelmistokehitystä tukevaa dokumentaatiota käsiteltiin luvussa 3.3.1.

### 4.4.2 Ohjelmistokehityksen kustannukset ja tuotot

Tutkittavan yrityksen nykykäytännöillä, ohjelmistokehitys jää suurilta osin käyttöönottovaiheeseen, joten ohjelmistokehityksen todellisia kustannuksia ei ole helppo arvioida. Asiakkaan luona työtuntia kohden laskettu kehityskustannus on moninkertainen verrattuna toimistolla tapahtuvaan kehitykseen. Tuntikustannuksen suuruus riippuu muun muassa käyttöönotettavan laitoksen maantieteellisestä sijainnista. Projektiorganisaation edustajan karkean arvion mukaan, ohjelmistonkehitykseen (ennen käyttöönottoa) käytetään aikaa keskimäärin 3–6 henkilötyökuukautta. Käyttöönottoon kuluu keskimäärin 2 henkilötyökuukautta. Tuotekehitysorganisaation edustajan mukaan ohjelmointityö jää yhä enenevissä määrin käyttöönottoon. Käyttöönotto siis pitkittyy ohjelmiston matalan valmiusasteen vuoksi.

Yrityksellä ei myöskään ole sisäisen testauksen menettelyjä tai FAT-testejä. Ohjelmiston keskeneräisyys siis huomataan vasta käyttöönottovaiheessa. Tuotekehitysorganisaation

tulisi kehittää FAT-testausmenettely, jotta puutteet ja ohjelmointivirheet tunnistettaisiin ajoissa. Tällöin tunnistetut virheet ja puutteet voitaisiin korjata, ennen varsinaista käyttöönottoa ja SAT-testejä.

Automaatio-ohjelmistojen kustannuksia voidaan arvioida I/O-pisteiden lukumäärän mukaan. I/O-pisteiden lukumäärä kuvaa antureiden ja toimilaitteiden yhteenlaskettua määrää. Yhdysvaltalaisen automaatio-ohjelmistointegraattorin [57] mukaan ohjelmistokehitykseen, ennen käyttöönottoa, kuluu aikaa keskimäärin 3.5 tuntia I/O-pistettä kohti. Arvioon sisältyy oletus, jonka mukaan projektin koko on noin 150–200 I/O-pistettä. Ohjelmistokehityksen työmäärä (tunteina I/O-pistettä kohti) jakautuu tässä arviossa taulukon 8 mukaisesti.

***Taulukko 8.*** *Ohjelmistokehityksen työmäärä tunteina.*

| **Ohjelmistokehityksen vaihe** | **Työmäärä (tuntia)** |
|---|---|
| Toiminnalliset vaatimukset ja suunnittelu | 1.0 |
| Ohjelmointi, sisäinen testaus ja integrointi | 2.0 |
| Tehdashyväksyntätestit (FAT) | 0.5 |
| Yhteensä | 3.5 |

Tapaustutkimuksen tuotantolinjalla on sähkökomponenttiluettelon mukaan yhteensä 147 I/O-pistettä, joista 97 liittyy murskaimeen. Murskain on siis huomattavasti muita prosessilaitteita kompleksisempi. Kompleksisuuden takia, murskaimen toiminnoissa on eniten kehitettävää ja testattavaa. Kustannusestimaatit onkin laadittu erikseen murskaimen ohjelmistolle ja koko tuotantolinjan ohjelmistolle. Näin voidaan vertailla kehityskustannuksien keskinäisiä suhteita. Yksittäisen ohjelmiston kehityskustannus voidaan esittää yhtälöllä,

$$kehityskustannus = tuntikustannus \times työmäärä \times IOpisteet\ . \qquad (5)$$

Nykytilanteessa ohjelmistot teetetään alihankintana. joten kustannuslaskelmissa ohjelmistokehityksen tuntikustannukseksi on oletettu 100 euroa. Kustannuslaskelman pääasiallinen tarkoitus on arvioida ohjelmistokehityksen kokonaistyömäärää ja työmäärästä aiheutuvia kustannuksia. Tuntikustannuksen suuruus ei vaikuta työmäärien vertailukel-

poisuuteen. Kuten yhtälöstä (5) nähdään, ohjelmiston kehityksenkustannuksen ja tuntikustannuksen välinen yhteys on lineaarinen. Työskentelyyn liittyvät oletukset on kuvattu taulukossa 9.

***Taulukko 9.*** *Työaika- ja kustannusparametrit.*

| | |
|---|---|
| **Työpäivien lukumäärä kuukaudessa (kpl)** | 20 |
| **Työpäivän pituus (tuntia)** | 7.5 |
| **Työn tuntihinta (euroa)** | 100 |
| **Ohjelmistokehitykseen osuus työajasta (%)** | 100 |

Tapaustutkimuksen tuotantolinjaa voidaan pitää suppeana projektina, sillä kyseisessä tuotantolinjassa ei käytetä magneettierottimen lisäksi muita erottelulaitteita, kuten pyörrevirtaerotinta tai ilmaluokittelijaa. Tapaustutkimukseen liittyvällä laitoksella ei myöskään ole rinnakkaisia SRF-tuotantolinjoja. Taulukolla 8 ja yhtälöllä (5) lasketun FAT-testatun tuotantolinjan kehityskustannuksia voidaan verrata suppeaan 3 kuukautta kestävään ohjelmistoprojektiin. Nykykäytännöillä tuotetun ohjelmistoprojektin kehityskustannuksia on arvioitu käyttäen taulukon 9 arvoja. Kustannuslaskelmat on esitetty liitteessä E.

***Taulukko 10.*** Ohjelmistojen kehityskustannukset ennen käyttöönottovaihetta.

| | |
|---|---|
| **Suppean ohjelmistoprojektin kehityskustannukset nykykäytännöillä (kesto 3 kuukautta)** | 45 tuhatta euroa |
| **Tuotantolinjalle kehitettävän ohjelmiston kustannukset (FAT-testauksella)** | 51 tuhatta euroa |
| **Murskaimelle kehitettävän ohjelmiston kustannukset (FAT-testauksella)** | 34 tuhatta euroa |

Taulukon 10 perusteella huomataan, että FAT-testauksella ohjelmistokehityksen kustannukset ovat noin 13 % suuremmat kuin ilman FAT-testiä. Taulukon 9 perusteella havaitaan, että FAT-testauksen osuus ohjelmistokehityksestä on noin 14 prosenttia. I/O-pisteiden lukumäärän mukaan tehtävän kustannusestimaatin voidaan siis todeta tuottavan oikean suuntaisia tuloksia.

Huomionarvoista on, että nykykäytäntöjen mukaisen ei-testatun ohjelmiston kehityskustannukset ovat huomattavasti taulukon 10 arvoa suuremmat. Tämä johtuu siitä, että merkittävä osa ohjelmistokehityksestä ja virheenkorjauksista tapahtuu tällöin käyttöönotossa. FAT-testauksen voidaan siis olettaa tuovan huomattavia kustannussäästöjä.
Taulukon 8 ohjelmistokehityksen vaiheet vastaavat luvussa 2.3.3 esitettyä automaatiosovelluksen kehityksen elinkaarimallia, jossa varsinaisen ohjelmointityön osuus on vain noin puolet kokonaiskehitysajasta. Ensimmäisessä vaiheessa määritettyjä toiminnallisia vaatimuksia verrataan FAT-testissä saataviin tuloksiin. Tällaisen systemaattisen ohjelmistotuotantoprosessin avulla voidaan kehitystyölle luoda selkeä aikataulu. Aikataulun avulla voidaan seurata, pysyykö ohjelmistoprojekti kustannus- ja aikataulutavoitteissaan. Liitteen E kustannusarviot toimivat siis sekä kustannusten kohdennuksen että projektin aikataulutuksen pohjana.
Investoinnin kannattavuutta voidaan mitata takaisinmaksuajalla. Takaisinmaksuaika tarkoittaa ajanhetkeä, jonka jälkeen investointiin liittyvä kassavirta muuttuu positiiviseksi. Takaisinmaksuaika kuvaa siis investoinnin break-even -pisteen saavuttamiseen kuluvaa aikaa. Tyypillisesti takaisinmaksuajan yksikkönä käytetään vuosia. Takaisinmaksuaika voidaan esittää seuraavasti

$$takaisinmaksuaika = investoinnin\ kustannukset / vuotuiset\ tuotot\ . \qquad (6)$$

Kuten luvussa 2.2.4 todettiin, modulaaristen ohjelmistotuoteperheiden kehitykseen vaaditun alkuinvestoinnin voidaan olettaa olevan kolminkertainen suhteessa yksittäiselle systeemille kehitettävän ohjelmiston kehityskustannuksiin. Luvussa myös mainittiin, että ohjelmistotuoteperheitä hyödyntämällä, yksittäisen ohjelmiston kehitykseen kuluva työmäärä vähenee alle puoleen verrattuna ei-modulaariseen ohjelmistoon.
Yhtälön (6) investoinnin kustannukset ja vuotuiset tuotot ovat suoraan verrannollisia yksittäiselle systeemille kehitettävän ohjelmiston kehityskustannuksiin. Siirtymäprosessin takaisinmaksuaika voidaankin esittää seuraavasti:

$$takaisinmaksuaika = \frac{alkuinvestointikerroin}{työmääräkerroin} \times (ohjelmistoprojektien\ määrä). \qquad (7)$$

Alkuinvestointikerroin kuvaa moduulien rakentamiseen vaadittavaa työmäärää suhteessa ei-modulaarisen ohjelmiston kehityskustannuksiin. Työmääräkerroin kuvaa uudelleenkäytettävien moduulien avulla rakennetun ohjelmiston kehitysaikaa suhteessa ei-modulaarisen ohjelmiston kehitysaikaan. Ohjelmistoprojektien määrä on vuoden aikana

projekteille kehitettävien ohjelmistojen lukumäärä. Huomionarvoista on, ettei tuntikustannuksen suuruus siis vaikuta takaisinmaksuaikaan.

Projektille tuotettavien ohjelmistojen kehitysaika on alle 12 kuukautta. Liitteen E investointilaskelmassa on oletettu, että kehitettyjen ohjelmistojen vuotuinen lukumäärä vastaa tiettynä vuotena toimitettujen projektien määrää. Referenssitietona on käytetty vuonna 2016 toimitettujen murskainten ja kokonaisten tuotantolinjojen lukumäärää. Investointilaskelmaan liittyvät oletukset on esitetty taulukossa 11.

***Taulukko 11.*** *Investointilaskelman parametrit.*

| **Parametri** | **Murskain** | **Tuotantolinja** |
|---|---:|---:|
| Alkuinvestointikerroin | 3.0 | 3.0 |
| Työmääräkerroin | 0.5 | 0.5 |
| Toimitusten lukumäärä | 7 | 6 |

Taulukon 10 ja yhtälön (7) perusteella, modulaarisen tuotantolinjan kaikille tuotteille (askelsyötin, murskain, poistokuljetin ja magneettierotin) kehitettävien tuoteperheiden vaatiman investoinnin takaisinmaksuajaksi on arvioitu yksi vuosi. Pelkän murskaimen modulaarisen tuoteperheen vaatiman investoinnin takaisinmaksuaika on 0.9 vuotta eli noin 11 kuukautta. Investointilaskelmat on esitelty liitteessä E. Takaisinmaksuajassa ei ole huomioitu kuvan 43 työpajojen kustannuksia tai lisäinvestointeja mallipohjaisen ohjelmistokehityksen työkaluihin.

Modulaaristen tuoteperheiden Investointikustannukset on laskettu taulukon 10 FAT-testatun ohjelmiston kustannusten ja taulukon 11 alkuinvestointikertoimen välisenä tulona. Liitteessä E on lisäksi esitelty lisäinvestointiehdotukset työpajoihin ja mallipohjaisen ohjelmistokehityksen työkaluihin. Investointikustannukset on esitelty taulukossa 12.

***Taulukko 12.*** *Investointikustannukset.*

| **Investointi** | **Kustannus (tuhatta euroa)** |
|---|---|
| Murskaimen moduulit | 102 |
| Koko tuotantolinjan moduulit | 153 |
| Työkalusetti 1 | 12 |
| Työkalusetti 2 | 13 |
| Työpajat | 4 |

Taulukon 12 työkalusetit ovat keskenään vaihtoehtoisia. Molemmat sisältävät MathWorksin Stateflow-työkalun, jolla voidaan mallintaa ohjauslogiikkaa tilakoneiden ja vuokaavioiden avulla. Stateflow soveltuu esimerkiksi boolean-logiikalla totutettujen lukitusketjujen mallinnukseen ja vikaantumistilanteiden analysointiin. Stateflow tukee myös Simulinkin automaattista koodin generointia. Työkaluseteissä on mukana myös Simulink-malleja ja Stateflow-kaavioita hyödyntävät PLC-koodigeneraattorit.

Työkalusetti 1 sisältää Simulink PLC Coder –työkalun, joka generoi standardin IEC 61131 mukaista structured text –lähdekoodia Simulink-malleista. Työkalu tukee Siemensin S7-logiikkaohjaimien lisäksi useita muita PLC-alustoja. Työkalusetissä 2 on Siemensin kehittämä Target 1500S –työkalu koodin generointiin. Työkalu tukee Siemensin S7-1500 logiikkaohjainalustaa. Target 1500S muuntaa MATLAB- ja Simulink Coder –työkalujen tuottamat C-kieliset koodit Step 7 yhteensopivaksi logiikkakoodiksi. Toisin kuin Simulink PLC Coder, Siemensin Target 1500S- koodigeneraattori vaatii myös MathWorksin Simulink ja MATLAB Coder-työkalut. Ratkaisun hinta siis nousee MathWorksin tarjoamaa PLC-koodigeneraattoriratkaisua korkeammaksi. MathWorksin PLC Coderilla on myös laajempi PLC-alustatuki. PLC Coder tukee kaikkia S7-sarjan PLC-alustoja. Lisäksi PLC Coder tukee esimerkiksi Rockwell Automationin ja Omronin PLC-alustoja. PLC Coder –työkalu on parempi valinta, koska se ei ole riippuvainen tietyn valmistajan PLC-alustasta.

Mallipohjaisella suunnittelulla ja automaattisella koodin generoinnilla voidaan vähentää ohjelmointityön määrää ja luoda tarkempaa dokumentaatiota vähemmällä vaivalla sekä tehdä nopeasti suuria muutoksia koodiin, simulointimallin ollessa olemassa. Kuten taulukosta 12 nähdään koodigeneraattorien lisenssien hinnat ovat kuitenkin suhteellisen korkeita. Uusien ohjelmistokehitysmenetelmien ja työkalujen opettelu vie myös paljon aikaa. Koodigeneraattoreista on hyötyä vain kompleksisten algoritmien ja toimintojen implementoinnissa. Tällaisia kompleksisia tehtäviä ovat esimerkiksi dynaaminen säätö ja

vaativat turva-automaatiotoiminnot. Koodigeneraattorin hyötynä voidaan pitää myös standardoitua ohjelmointityyliä. Simulinkin visuaaliset lohkokaavioesitykset myös parantavat ohjelmiston toimintojen ymmärrettävyyttä ja helpottavat ylläpitoa. Dokumentoituja malleja voidaan käyttää tuotekehityksen ja implementointitiimin välisessä kommunikoinnissa. Mikäli ohjelmistokehitys on ulkoistettu alihankkijoille, teknisten yksityiskohtien täsmällinen esittäminen on erityisen tärkeää. Lisäksi koodin uudelleenkäyttö helpottuu, sillä Simulink-kirjastojen lohkot ovat modulaarisia.

Automaattisesta koodin generoinnista olisi hyötyä erityisesti murskaimen tapauksessa, jonka toiminnot ovat kompleksisia. Murskaimen toimintojen dokumentaatiokin vaatii tarkennusta. Koodin generointia voitaisiin käyttää myös muiden laitteiden suorituskykyparannusten implementointiin.

Työkaluinvestoinnit ovat noin kertaluokan pienempiä kuin moduulien luomiseen vaadittavat investoinnit. Mallipohjaiset menetelmät voisivat siis tarjota kustannustehokkaamman tavan kehittää ohjelmistojen modulaarisuutta. Toimintojen dokumentoinnin ja koodin implementoinnin lisäksi mallipohjainen suunnittelu tarjoaa keinon tutkia tuotevarianttien vaikutuksia automaatiotoimintoihin. 3D-simuloinnilla voidaan tutkia eri laitosten murskaimia ja niiden mekaniikkamalleja sekä ohjelmiston tai ohjelmistomoduulin yhteensopivuutta eri laitevarianttien kanssa. Simulointia voidaan siis käyttää kuvan 43 mukaisesti laitevarianttien ja koodivarianttien analysointiin sekä tuoteperheiden suunnitteluun. Mallipohjaisella suunnittelulla voidaan myös lyhentää tuotteen suunnitteluaikaa, rinnakkaisen suunnittelun avulla. Tällä hetkellä, prosessilaitteen suunnittelu etenee sekventiaalisesti. Ensin laite suunnitellaan mekaanisesti, jonka jälkeen se myydään asiakkaalle. Tämän jälkeen valitaan sähköautomaatiolaitteisto (anturit ja toimilaitteet) ja suunnitellaan laitteen automaatiotoiminnot. Varsinainen ohjelmistokehitys alkaa vasta sähköautomaatiosuunnittelun päätyttyä. Näin ollen ohjelmistokehitykseen jää vain vähän aikaa verrattuna laitteistosuunnitteluun.

Rinnakkaisen suunnittelun tarkoituksena on mahdollistaa samanaikainen työskentely eri suunnitteluvaiheissa. Liitteen F työnkulku kuvaa menettelytapaa, joka mahdollistaa mallipohjaisen ohjelmistokehityksen aloittamisen jo tuotekehitysprojektin varhaisessa vaiheessa. Tämä perustuu luvussa 4.2 kuvattuihin abstraktiotasoihin. Lähestymistavassa automaatiotoimintojen ja ohjauslogiikan suunnittelu voidaan aloittaa laitteistosuunnittelun rinnalla abstrakteilla malleilla. Laitteistosuunnittelun tuotosten tarkentuessa, myös ohjelmiston suunnitteluun käytettävää mallia tarkennetaan. Tällöin ohjelmiston suunnitteluun jää enemmän aikaa. Menettely mahdollistaa myös jatkuvan testaamisen ja validoinnin läpi tuotekehityksen elinkaaren. Liitteen F työnkulun suunnittelussa on mukailtu Siemensin Mechatronics Concept Designer –työkalun lähestymistapaa, perustuen lähteeseen [58].

Liitteestä E nähdään, että murskaimen ohjelmiston modularisoinnilla on lyhyempi takaisinmaksuaika, kuin koko tuotantolinjan ohjelmiston modularisoinnilla. Syynä tähän on se, että murskaimia toimitetaan myös erikseen. Murskaus on myös yrityksen ydinosaamisalue, joten murskaimien tuotekehitykseen on syytä panostaa muita tuotteita enemmän.

Kuten luvussa 2.2.4 todettiin, siirtymä kannattaa suorittaa pienin askelin. Modulaarista ohjelmistoa voidaan lähteä aluksi kehittämään yksittäiselle murskaimelle. Modularisointityö voidaan edelleen jakaa pienempiin osatehtäviin, joissa kehitetään moduulit vain määritellyille murskaimen toiminnoille. Kuvan 37 mukaisesti, moduulikirjastojen rakentaminen kannattaa aloittaa yleisimmin käytetyistä kenttälaitteista. Tällöin moduuleja voidaan kehittää yksittäisten venttiilien ja moottorien ohjaukseen sekä pinnanmittausanturin ja induktiivisten rajakytkimien signaalinkäsittelyyn. Näin voidaan vähentää siirtymäprosessin sitomia resursseja ja vaadittavan alkuinvestoinnin määrää.

Modulaariset ohjelmistot mahdollistavat uusien ominaisuuksien lisäämisen ohjelmistoihin. Modulaarisuus siis tukee ohjelmistoliiketoimintaa. Asiakkaille voidaan siis tarjota uusia toiminnallisuuksia ja suorituskykyä lisääviä ohjelmistopäivityksiä. Luvussa 4.2.2 esitetty dynaaminen nopeuden säätö on yksi esimerkki tuotantolinjan suorituskykyisyyden parantamisesta, uusien toimintojen avulla. Ohjelmistopäivitysten hinnoittelu voi perustua, esimerkiksi asiakkaalle tuotettuun taloudelliseen hyötyyn. Päivitysten liiketoimintavaikutuksia voidaan arvioida esimerkiksi simuloimalla ja laitosten välisillä vertailuanalyyseillä. Ohjelmistojen lisäksi voidaan myydä myös uusia toiminnallisuuksia mahdollistavia antureita. Päivityksiä voidaan myydä paketteina, jotka sisältävät sekä uusia laitteistokomponentteja että ohjelmistopäivityksiä. Lisäksi huomionarvoista on, että voimalaitosten keskimääräinen tekninen elinikä on noin 40 vuotta. Sähköautomaatiokomponenttien elinikä taas on alle 15 vuotta, joten laitoksille voidaan myydä myös varaosia ja modernisointipalveluja.

Modulaarisen ohjelmiston kokonaiskustannus muodostuu alkuinvestoinnin lisäksi esimerkiksi koulutus- ja ylläpitokustannuksista. Liitteen E kustannusarviot saattavat siis olla liian optimistisia.

### 4.4.3 Kustannusten arviointiin liittyvät epävarmuustekijät

Edellisessä luvussa kuvattu kustannusten arvioinnin lähestymistapa perustuu ohjelmistokehityksen työmäärän estimointiin. Työmäärää on arvioitu kehitykseen käytettävien työtuntien avulla. Lähestymistavassa siis yhdistyy projektien aikataulutus ja kustannusten hallinta.

Kustannusarvio ei sisällä ohjelmistojen ja toimintojen dokumentointiin tarvittavaa työmäärää. Dokumentoinnin työmäärä vähenisi mallipohjaisen suunnittelun työkaluja hyödyntämällä. Tällä hetkellä toiminnankuvausten laatiminen jää projektiorganisaation tehtäväksi. Toiminnankuvausten ja muiden (kuvassa 32 esitettyjen) teknisten dokumenttien laadinta tulisi kuitenkin olla tuotekehityksen vastuulla. Tällöin asiakkaalle voitaisiin myydä valmiita tuotteita, eikä prosessilaitteita tarvitsisi kehittää projektikohtaisesti. Tämä tukee tuotteiden standardointia sekä projektin kustannusten ja aikataulun hallintaa.

Työmäärän arvioinnin suurin epävarmuustekijä on tuottavuuden estimointi. Tuottavuus kuvaa työn aikaansaannoksia suhteessa siihen käytettyyn aikaan. Ohjelmistokehittäjien tuottavuuteen vaikuttavat muun muassa motivaatio, kokemus ja tietotaito. Aikataulupaineet tyypillisesti lisäävät tuottavuutta, mutta saattavat vaikuttaa negatiivisesti ohjelmiston laatuun. Heikon laadun seurauksen ohjelmistojen ylläpidettävyys ja laajennettavuus kärsivät. Aikataulupaineet lisäävät myös ohjelmointivirheiden todennäköisyyttä. Manuaalisesta ohjelmointityöstä aiheutuvat virheet voidaan eliminoida automaattisella koodin generoinnilla.

Työmäärän arviointi perustui tässä tutkimuksessa I/O-pisteiden lukumäärään. I/O-pisteiden lukumäärä ei kuitenkaan yksiselitteisesti kuvaa ohjelmiston kompleksisuutta tai toimintojen määrää. Kompleksisuus lisää moduulien rakentamisen työmäärää. Toiminnallisuuksien määrä vaikuttaa moduulien määrän ja laajuuden valintaan.

Kustannuslaskelmissa on tarkasteltu vain kehityskustannuksia. Laadun ja tuotehallinnan parannuksista syntyvät kustannussäästöt on jätetty huomioimatta eikä ylläpitokustannuksiin ole otettu kantaa. Modulaariset ohjelmistot vähentävät myös ylläpidon työmäärää ja kustannuksia.

Tapaustutkimuksen perusteella, ehdotetaan siirtymistä mallipohjaiseen systeemisuunnitteluun ja ohjelmistokehitykseen. Kustannusarviossa ei kuitenkaan ole huomioitu uusien menetelmien ja työkalun oppimiseen ja ohjeistukseen vaadittavaa työmäärää. Uusien ohjelmistokehityksen menetelmien aiheuttamat viiveet ja kustannukset riippuvat oppimiskäyrän jyrkkyydestä. MathWorksin ohjelmointiympäristön etuna on kattava ja huolellisesti laadittu kehitystyökalujen tukidokumentaatio. Tämä edistää uusien menetelmien omaksumista ja työkalujen tehokasta käyttöönottoa.

Samalta toimialalta ei myöskään löydy vastaavia ohjelmistojen modularisointiin liittyviä projekteja, joita voisi käyttää vertailupohjana. Tässä tutkimuksessa esitetty kustannusestimaatti on siis vain suuntaa antava. Estimaattia on siis tarkennettava siirtymäprosessin edetessä. Taulukossa 8 esitetyt työmäärää kuvaavat parametrit perustuvat automaatioalan ohjelmistointegraattorin käytännönkokemuksiin, joten niitä voidaan pitää hyvänä perustana ensimmäisen kustannusestimaatin laatimiseen. Taulukon 11 modularisoinnin työmäärää kuvaavat parametrit perustuvat ohjelmistoalan tapaustutkimuksiin. Kuten

edellisessä luvussa todettiin, työn tuntikustannus ei vaikuta investoinnille laskettuun takaisinmaksuaikaan. Liitteen E investointilaskelmaa voidaan siis pitää käyttökelpoisena kannattavuuden arviointimenetelmänä, sillä se on laadittu käyttäen parasta saavilla olevaa informaatiota.

Murskaimen takaisinmaksuajaksi saatiin liitteessä E 0.9 vuotta. Investointia, jonka takaisinmaksuaika on alle vuosi, pidetään yleensä erittäin kannattavana. Tapaustutkimuksen havaintojen pohjalta, suositellaan siirtymäprosessin aloittamista murskaimeen liittyvän ohjelmistototeutuksen modularisoinnista. Siirtymäprosessi kannattaa tehdä pienin askelin, aloittaen yksittäisiin antureihin ja toimilaitteisiin liittyvien moduulikirjastojen laatimisesta. Liitteessä G on esitelty ohjelmistotuotannon tärkeimmät kehityskohteet tutkimuksen kohteena olleessa yrityksessä.

# 5. YHTEENVETO

Tutkimuksen tavoitteena oli selvittää, miten modulaarisuus ja versionhallinta tukevat ohjelmistokehitystä ja yrityksen liiketoimintaa. Tutkimuksen kohteena olleelle tuotantolinjalle löydettiin kuvailutapa, jossa osasysteemien väliset rajapinnat kuvattiin materiaalin ja informaation virtauksena. Nämä rajapinnat mahdollistavat tuotantolinjan modulaarisen rakenteen. Tällöin muutokset yhdessä osasysteemissä eivät vaikuta muihin osasysteemeihin, jos rajapinnat säilyvät ennallaan. Kuvailutavan käyttötarkoituksena on tukea simulaattorikehitystä. Simulaatioita voidaan hyödyntää ohjelmistojen kehittämisessä ja testaamisessa. Kuvailutavan yleispätevyyttä ei kuitenkaan tässä tutkimuksessa todistettu. Kuvailutavan yleispätevyys ja soveltuvuus simulaattorikehitykseen tulee testata luomalla simulointimalleja tuotantolinjavarianteista, tutkimuksessa kuvailluilla abstraktiotasoilla. Kuvaillun tuotetiedon muuntaminen dynaamiseksi simulointimalliksi ei myöskään sisältynyt tähän tutkimukseen, vaan käytännön toteutus vaatii vielä jatkotutkimusta. Jatkotutkimusaiheeksi ehdotetaan tiedonvaihtomenetelmän selvittämistä, jolla versionhallinnan tuotetiedosta voidaan luoda dynaaminen simulointimalli. Tällä hetkellä yksittäisen simulointimallin luominen vaatii paljon manuaalista implementointityötä. Simulointimallien luominen on siis aikaa vievä ja virhealtis aktiviteetti. Myös tuotetiedosta generoitujen simulointimallien ajantasaisuuden varmistaminen vaatii lisätutkimusta. Versionhallinta- ja simulointiympäristön välinen tiedonvaihto tulee siis toteuttaa siten, että tuotetiedon muuttuessa vastaavat muutokset päivittyvät myös käytettyihin simulointimalleihin.
Versionhallinnalle määritettiin tavoitteet ja vaatimukset sekä yrityksen tarpeisiin soveltuva työkalu versionhallinnan toteuttamiseksi. Ratkaisussa ohjelmistojen ja simulointimallien versionhallinta integroidaan osaksi tuotantolinjalle määriteltyä tuotetietorakennetta. Tuotantolinjan tuotetiedon kokonaisvaltainen kuvaaminen integroidulla tuotetietomallilla, yrityksen PDM-työkalua hyödyntäen, osoittautui toteuttamiskelpoiseksi ratkaisuksi. Versionhallinnan implementointi ei kuitenkaan sisältynyt tutkimukseen. Implementoinnin jälkeisten käyttökokemuksien perusteella tulee siis arvioida, onko PDM-työkalun käyttäminen ohjelmistojen ja simulointimallien versionhallintaan suotuisa vaihtoehto verrattuna muihin saatavissa oleviin työkaluihin, kuten Subversion tai versiondog. Lisätutkimusta vaatii erityisesti mahdollisuus hallita PLC-ohjelmistoprojekteja siten, että versionhallintaan vietäisiin ZIP-pakettien sijaan ainoastaan ohjelmistoissa tapahtuneet varsinaiset muutokset. Ratkaisu tähän ongelmaan voisi olla ohjelmiston modularisointi siten, että projektipohjaan ei tehdä muutoksia vaan ainoastaan moduulikirjastoihin. Tällä hetkellä,

yrityksen tuottamat PLC-ohjelmistot ovat kuitenkin ei-modulaarisia. Ohjelmistojen modulaariseen rakenteeseen liittyen saavutettiin vain abstrakteja tuloksia. Valitun versionhallintatyökalun tukea ehdotetulle modularisoinnin lähestymistavalle ei siis ole todistettu tai testattu. Modulaaristen ohjelmistojen versionhallintaa vaikeuttaa ainakin sulauttamistoiminnon puuttuminen PDM-työkalusta.

Siirtyminen modulaarisiin ohjelmistoihin arvioitiin taloudellisesti erittäin kannattavaksi. Siirtymän vaatiman investoinnin kannattavuutta arvioitiin takaisinmaksuaikaan perustuvalla menetelmällä. Investoinnin kustannusten ja takaisinmaksuajan estimaatteihin liittyy kuitenkin monia epävarmuustekijöitä. Laadittujen estimaattien tarkkuutta rajoittaa erityisesti vertailupohjan puuttuminen, joten estimaattia tulisi päivittää siirtymäprosessin edetessä.

Modulaaristen ohjelmistojen kehittämiseksi ehdotettiin myös mallipohjaisten suunnittelumenetelmien käyttöönottoa. Mallipohjaisten menetelmien eduiksi tunnistettiin kehitettävien ohjelmistojen jatkuva testaaminen, toimintojen dokumentoinnin helppous ja mallien hyödyntäminen sidosryhmien, kuten alihankkijoiden, välisessä kommunikoinnissa. Menetelmien hyödyntämisen haasteina ovat standardoidun kehitysalustan puuttuminen sekä laite- ja laitosvarianttien suuri määrä. Uusien ohjelmistokehitysmenetelmien käyttöönottaminen vaatii tietotaidon ja resurssien lisäämistä organisaation sisällä.

# LÄHTEET

[1] ISO/IEC/IEEE International Standard – Systems and software engineering – Vocabulary, in: ISO/IEC/IEEE 24765:2017(E), 2017, pp. 1-541.

[2] C. Lindkvist, A. Stasis, J. Whyte, Configuration Management in Complex Engineering Projects, Procedia CIRP, Vol. 11, 2013, pp. 173-176.

[3] ISO 10007:2018, Quality management – Guidelines for configuration management, 2018, pp. 1-14.

[4] IEEE Standard for Configuration Management in Systems and Software Engineering, in: IEEE Std 828-2012 (Revision of IEEE Std 828-2005), 2012, pp. 1-71.

[5] J. Conrad, T. Deubel, C. Köhler, S. Wanke, C. Weber, Change impact and risk analysis (CIRA) – combining the CPM/PDD theory and FMEA-methodology for an improved engineering change management, International Conference on Engineering Design ICED'07, 2007, pp. 1-11.

[6] J. Estublier, D. Leblang, A.v.d. Hoek, R. Conradi, G. Clemm, W. Tichy, D. Wiborg-Weber, Impact of software engineering research on the practice of software configuration management, ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 14, Iss. 4, 2005, pp. 383-430.

[7] L. Bendix, L. Borracci, Towards a suite of software configuration management metrics, Proceedings of the 12th international workshop on software configuration management, ACM, 2005, pp. 75-82.

[8] R.S. Pressman, Software engineering: a practitioner's approach, 5.th ed. McGraw-Hill, Boston, 2001, pp. 588-590.

[9] A. Abran, J.W. Moore, I. Books24x7, Guide to the software engineering body of knowledge: 2004 version: SWEBOK, IEEE Computer Society Press, Los Alamitos, Calif, 2004, pp. 124.

[10] R. Conradi, B. Westfechtel, Version models for software configuration management, ACM Computing Surveys (CSUR), Vol. 30, Iss. 2, 1998, pp. 232-282.

[11] B. O'Sullivan, Making sense of revision-control systems, Communications of the ACM, Vol. 52, Iss. 9, 2009, pp. 62.

[12] B. Vogel-Heuser, J. Fischer, S. Rosch, S. Feldmann, S. Ulewicz, Challenges for maintenance of PLC-software and its related hardware for automated production systems: Selected industrial Case Studies, IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2015, pp. 362-371.

[13] D. Jenkins, J. Arnaud, S. Thompson, M. Yau, J. Wright, Version Control and Patch Management of Protection and Automation Systems, 12th IET International Conference on Developments in Power System Protection (DPSP), IET, 2014, pp. 12-80.

[14] A. Buda, P. Makkonen, R. Derroisne, V. Cheutet, PDM suitability study for CAE data management, PLM11 8th International Conference on Product Lifecycle Management, 2011, pp. 229-238.

[15] G. Walker, J. Friedman, R. Aberg, Configuration Management of the Model-Based Design Process, verkkodokumentti, The MathWorks, Natick, 2017, pp. 1-8. Haettu (2.8.2019), Saatavissa: https://se.mathworks.com/content/dam/mathworks/tag-team/Objects/s/40503_SAE-2007-01-1775-Configuration-Management-MathWorks.pdf.pdf

[16] P. Novák, E. Serral, R. Mordinyi, R. Šindelář, Integrating heterogeneous engineering knowledge and tools for efficient industrial simulation model support, Advanced Engineering Informatics, Vol. 29, Iss. 3, 2015, pp. 575-590.

[17] F. Linden, E. Rommes, K. Schmid, Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering, 1. Aufl. ed. Springer-Verlag, Berlin, Heidelberg, 2007, pp. 3-20.

[18] H.P. Breivold, S. Larsson, R. Land, Migrating Industrial Systems towards Software Product Lines: Experiences and Observations through Case Studies, 2008 34th Euromicro Conference Software Engineering and Advanced Applications, IEEE, pp. 232-239.

[19] J. Fischer, B. Vogel-Heuser, D. Friedrich, Configuration of PLC software for automated warehouses based on reusable components- an industrial case study, IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA), IEEE, 2015, pp. 1-7.

[20] ISO/IEC/IEEE International Standard – Systems and software engineering – Life cycle management – Part 1: Guidelines for life cycle management, in: ISO/IEC/IEEE 24748-1:2018(E), 2018, pp. 1-82.

[21] B. Vogel-Heuser, A. Fay, I. Schaefer, M. Tichy, Evolution of software in automated production systems: Challenges and research directions, The Journal of Systems & Software, Vol. 110, 2015, pp. 54-84.

[22] A. Sääksvuori, A. Immonen, Product Lifecycle Management, 3. Aufl.; Third ed. Springer-Verlag, Berlin, Heidelberg, 2008, pp. 1-24.

[23] J. Li, F. Tao, Y. Cheng, L. Zhao, Big Data in product lifecycle management, The International Journal of Advanced Manufacturing Technology, Vol. 81, Iss. 1, 2015, pp. 667-684.

[24] T. Parkkila, J. Kääriäinen, H. Tanner, J. Riekki, Middle-of-life PLM solutions for reconfigurable networked mechatronic products, Journal of Systems and Software Vol.2 Nr.5, 177-187, 2012, pp. 177-186.

[25] A. Deuter, S. Rizzo, A Critical View on PLM/ALM Convergence in Practice and Research, Procedia Technology, Vol. 26, 2016, pp. 405-412.

[26] M. Bricogne, L. Rivest, N. Troussier, B. Eynard, Towards PLM for Mechatronics System Design Using Concurrent Software Versioning Principles, 9th International Conference on Product Lifecycle Management (PLM), 2012, pp. 339-348.

[27] K. Thramboulidis, IEC 61131 as enabler of OO and MDD in industrial automation, IEEE 10th International Conference on Industrial Informatics, IEEE, 2012, pp. 425-430.

[28] A. Dubey, Evaluating software engineering methods in the context of automation applications, 9th IEEE International Conference on Industrial Informatics, IEEE, 2011, pp. 585-590.

[29] IEC 61511-2, Functional safety – Safety instrumented systems for the process industry sector – Part 2: Guidelines for the application of IEC 61511-1, 2017, pp. 1-449.

[30] P. Nguyen, M. Kramer, J. Klein, Y. Le Traon, An Extensive Systematic Review on Model-Driven Development of Secure Systems, Information and Software Technology, Vol. 68, 2015, pp. 62-81.

[31] A. Reichwein, C. Paredis, Overview of Architecture Frameworks and Modeling Languages for Model-Based Systems Engineering, Proceedings of the ASME Design Engineering Technical Conference, Vol. 2, 2011, pp.1-9.

[32] S. Kleiner, C. Kramer, Model Based Design with Systems Engineering Based on RFLP Using V6, in: Smart Product Engineering: Proceedings of the 23rd CIRP Design Conference, Bochum, Germany, 2013, pp. 93-102.

[33] O. Graeser, B. Kumar, N. Moriz, A. Maier, O. Niggemann, AutomationML as a Basis for Offline- and Realtime-Simulation, 8th International Conference on Informatics in Control, Automation and Robotics (INIC), 2011, pp. 359-368.

[34] N. Schetinin, N. Moriz, B. Kumar, A. Maier, S. Faltinski, O. Niggemann, Why do verification approaches in automation rarely use HIL-test?, IEEE International Conference on Industrial Technology (ICIT), IEEE, 2013, pp. 1428-1433.

[35] R. Isermann, J. Schaffnit, S. Sinsel, Hardware-in-the-Loop Simulation for the Design and Testing of Engine-Control Systems, IFAC Proceedings Volumes, Vol. 31, Iss. 4, 1998, pp. 1-10.

[36] M. Follmer, P. Hehenberger, S. Punz, R. Rosen, K. Zeman, Approach for the creation of mechatronic system models, Vol. 4, 2011, pp. 258-267.

[37] O. Niggemann, J. Stroop, Models for model's sake, ACM/IEEE 30th International Conference on Software Engineering, IEEE, 2008, pp. 561-570.

[38] S. Faltinski, O. Niggemann, N. Moriz, A. Mankowski, AutomationML: From data exchange to system planning and simulation, IEEE International Conference on Industrial Technology, IEEE, 2012, pp. 378-383.

[39] P. Hehenberger, Perspectives on hierarchical modeling in mechatronic design, Advanced Engineering Informatics, Vol. 28, Iss. 3, 2014, pp. 188-197.

[40] O. Niggemann, System-level design and simulation of automation systems, IEEE International Workshop on Factory Communication Systems Proceedings, IEEE, 2010, pp. 173-176.

[41] T. Moser, R. Mordinyi, D. Winkler, Extending mechatronic objects for automation systems engineering in heterogeneous engineering environments, Proceedings of IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA), IEEE, 2012, pp. 1-8.

[42] R. Drath, A. Luder, J. Peschke, L. Hundt, AutomationML – the glue for seamless automation engineering, IEEE International Conference on Emerging Technologies and Factory Automation, IEEE, 2008, pp. 616-623.

[43] IEC 62708:2015 Document kinds for Electrical and Instrumentation Projects in the Process Industry, 2015, pp. 1-11.

[44] ISO 13849-1:2015, Safety of machinery – Safety-related parts of control systems – Part 1: General principles for design, 2015, pp. 1-86.

[45] IEC 62061:2005, Safety of machinery. Functional safety of safety-related electrical, electronic and programmable electronic control systems, 2005, pp. 1-205.

[46] X. Wu, A. Murray, M.-. Storey, R. Lintern, A reverse engineering approach to support software maintenance: version control knowledge extraction, 11th Working Conference on Reverse Engineering, IEEE, 2004, pp. 90-99.

[47] D. Bergsjö, J. Malmqvist, M. Ström, Implementing Support for Management of Mechatronic Product Data in PLM Systems: Two Case Studies, International Mechanical Engineering Congress and Exposition (ASME), 2006, pp. 1175-1183.

[48] IEC 61131-3:2013, Programmable controllers, part 3: Programming languages, 2013, pp. 1-235.

[49] H. Abid, P. Pernelle, C. Benamar, D. Noterman, A modelling approach of mechatronic products in PLM Systems, International Conference on Control, Decision and Information Technologies (CoDIT), 2013, pp. 714-718.

[50] H. Abid, P. Pernelle, C. Benamar, D. Noterman, Integration approach of mechatronics system in PLM systems, 19th International Conference on Automation and Computing, Chinese Automation and Computing Society in the UK (CACS), 2013, pp. 1-6.

[51] B.A. Talkhestani, N. Jazdi, W. Schloegl, M. Weyrich, Consistency check to synchronize the Digital Twin of manufacturing automation based on anchor points, Procedia CIRP, Vol. 72, 2018, pp. 159-164.

[52] Avfall Sverige, Swedish waste management, verkkodokumentti, Avfall Sverige AB, Malmö 2018, pp. 1-40. Haettu (1.8.2019), Saatavissa: https://www.avfallsverige.se/fileadmin/user_upload/Publikationer/Avfallshantering_2018_EN.pdf

[53] A. Papageorgiou, J.R. Barton, A. Karagiannidis, Assessment of the greenhouse effect impact of technologies used for energy recovery from municipal waste: A case for England, Journal of environmental management, Vol. 90, Iss. 10, 2009, pp. 2999-3012.

[54] VGB PowerTech, VGB-Standard KKS Pocketbook, verkkodokumentti, VGB PowerTech e.V., Essen 2019, pp. 1-148. Haettu (7.8.2019), Saatavissa: https://www.vgb.org/en/kks_pocketbook.html?dfid=96585

[55] S.G. McCrady, Tagnames and Signal Naming Conventions, in: Designing SCADA Application Software: A Practical Approach, 2013, pp. 58.

[56] Siemens, Integration of planning data from EPLAN Electric P8 to TIA Portal, verkkodokumentti, Siemens AG, Berliini 2017, pp. 1-20. Haettu (3.7.2019), Saatavissa: https://cache.industry.siemens.com/dl/files/224/109748224/att_933166/v1/109748224_EPLAN_to_TIA_Portal_DOC_v10_en.pdf

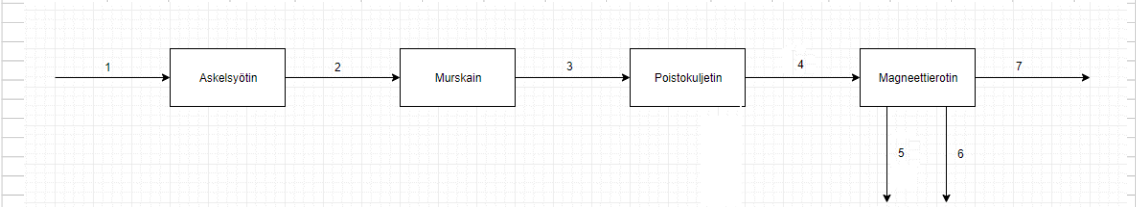
[57] J.R. Koelsch, Calculating the True Cost of Software, verkkodokumentti, Automation World, Chicago 2014, pp. 1-6. Haettu (27.6.2019), Saatavissa: https://www.automationworld.com/article/automation-strategies/design-engineering/optimization-and-validation/calculating-true-cost

[58] Siemens, Mechatronic Concept Designer, verkkodokumentti, Siemens PLM Software, Plano 2010, pp. 11-12. Haettu (6.7.2019), Saatavissa: http://www.alvarestech.com/temp/PDP2011/SiemensNXMechatronicsConceptDesigner/MCD_Quick_Start.pdf

# LIITE A: MASSATASELAKSELMA

| Massatase (SRF-tuotantolinja) | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Virtaus nro.** | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| **Komponentti** | | | | | | | |
| MIPS-hylky | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0 | 0 |
| Fe-metallit | 0.49 | 0.49 | 0.49 | 0.49 | 0 | 0.49 | 0 |
| käypä polttoaine | 8.26 | 8.26 | 8.26 | 8.26 | 0 | 0 | 8.26 |
| yhteensä (kg/s) | **9.72** | **9.72** | **9.72** | **9.72** | **0.97** | **0.49** | **8.26** |
| | | | | | | | |
| **TARKISTUS** | | | | | | | |
| 1 - 2 = | 0 | | | | | | |
| 2 -3 = | 0 | | | | | | |
| 3 - 4 = | 0 | | | | | | |
| 4 - 5 - 6 - 7 = | 0 | | | | | | |

| **RA Koostumus** | hylky | Fe | käypä |
|---|---|---|---|
| **Massaosuus** | 0.10 | 0.05 | 0.85 |
| **Määrä (kg/sek)** | 9.72 | | |
| **Tiheys (kg/m^3)** | 208 | | |




***Kuva A.1*** *Massataselaskelma.*

| **Device** | **Capacity (t/h)** | **Capacity (kg/s)** | **MaxVol (m^3)** | **MaxMass (kg)** | **Lenght (m)** | |
|---|---|---|---|---|---|---|
| Feeder | 60.00 | 16.67 | 75.00 | 15600.00 | 12.00 | |
| Shredder (for HHW) | 35.00 | 9.72 | | | | |
| **Device** | **Capacity (m^3/h)** | **Capacity (kg/s)** | **MaxSpeed (m/s)** | **Lenght (m)** | **travel time (s)** | **MaxMass (kg)** |
| Chain conveyor | 500.00 | 28.89 | 0.25 | 22.50 | 90.00 | 2600.00 |
| Belt conveyor | 500.00 | 28.89 | 1.00 | 4.00 | 4.00 | 115.56 |
| | | | | | | |
| **Conv: tons/hour to kg/sec** | | | | | | |
| 0.277777778 | | | | | | |

***Kuva A.2*** *Prosessilaitteiden ominaisuudet.*

# LIITE B: POISTOKULJETTIMEN RAJAPINNAT

**Poistokuljetin & PLC informaationvaihto (rajapintakuvaus kenttälaitetasolla - osa 1)**

| signaalin tunnus (TSNC) | signaalin nimitys (designation) | signaalin tyyppi | lähdemoduuli | lähdekomponentti | kohdemoduuli | kohdekomponentti |
|---|---|---|---|---|---|---|
| FPS_DCVR_IT_XI1 | Discharge Conveyor Current Trasmitter Indication | analoginen | poistokuljetin | virtamittari | poistokuljetin | PLC |
| FPS_DCVR_S1_XS1 | Discharge Conveyor Speed Switch 1 Status | binääri | poistokuljetin | pyörintävahti | poistokuljetin | PLC |
| FPS_DCVR_S2_XS2 | Discharge Conveyor Speed Switch 2 Status | binääri | poistokuljetin | pyörintävahti | poistokuljetin | PLC |
| FPS_DCVR_S3_XF1 | Discharge Conveyor Chocked Flow Switch Status | binääri | poistokuljetin | tukosvahti | poistokuljetin | PLC |
| | | | | | | |
| FPS_DCVR_M1_YC1 | Discharge Conveyor Motor Control ON/OFF Command | binääri | poistokuljetin | PLC | poistokuljetin | taajuusmuuttaja |
| FPS_DCVR_M1_YC2 | Discharge Conveyor Motor Control FW/BW Command | binääri | poistokuljetin | PLC | poistokuljetin | taajuusmuuttaja |
| | | | | | | |
| FPS_DCVR_M1_ZI1 | Discharge Conveyor Motor Status ON/OFF Input Flag | binääri | magneettierotin | PLC | poistokuljetin | PLC |
| FPS_DCVR_M1_ZI2 | Discharge Conveyor Motor Status FW/BW Input IFlag | binääri | magneettierotin | PLC | poistokuljetin | PLC |
| FPS_DCVR_FS_ZI3 | Discharge Conveyor Fault Status Input Flag | binääri | magneettierotin | PLC | poistokuljetin | PLC |
| FPS_DCVR_M1_ZO1 | Discharge Conveyor Motor Control ON/OFF Output Flag | binääri | poistokuljetin | PLC | murskain | PLC |
| FPS_DCVR_M1_ZO2 | Discharge Conveyor Motor Control FW/BW Output Flag | binääri | poistokuljetin | PLC | murskain | PLC |
| FPS_DCVR_FS_ZO3 | Discharge Conveyor Fault Status Output Flag | binääri | poistokuljetin | PLC | murskain | PLC |
| | | | | | | |
| FPS_MGNS_ES_XS1 | Magnet Separator Emergency Stop Cable Switch Status | binääri | magneettierotin | vaijerikytkin | magneettierotin | PLC |
| FPS_SHRD_R1_YC1 | Shredder Rotor Control ON/OFF Command | binääri | murskain | PLC | murskain | moottori |
| FPS_SHRD_R1_YC2 | Shredder Rotor Control FW/BW Command | binääri | murskain | PLC | murskain | moottori |

***Kuva B.1*** *Signaaliluettelo.*

**Poistokuljetin & PLC informaationvaihto (rajapintakuvaus kenttälaitetasolla - osa 2)**

| informaatio-komponentti | I/O-tyyppi | tieto-tyyppi | minimi-arvo | maksimi-arvo | kuvaus |
|---|---|---|---|---|---|
| virta | AI | Real | 4 | 20 | virtamittarin lähettimen virta-arvo (virtaviestinä 4...20 milliampeeria) |
| nopeus_1 | DI | Bool | 0 | 1 | pyörintävahdin 1 nopeuden tilatieto (1 = riittävä nopeus, 0 = alinopeus) |
| nopeus_2 | DI | Bool | 0 | 1 | pyörintävahdin 2 nopeuden tieto (1 = riittävä nopeus, 0 = alinopeus) |
| ei-tukosta | DI | Bool | 0 | 1 | tukosvahti tilatieto (1 = ei-tukosta, 0 = tukos havaittu) |
| | | | | | |
| käynnistä | DO | Bool | 0 | 1 | asettaa moottorin päälle tai pois päältä (1 = päälle, 0 = pois) |
| suuntaa | DO | Bool | 0 | 1 | asettaa liikkeen suunnan (1 = eteen, 0 = taakse) |
| | | | | | |
| käyntitieto | DI | Bool | 0 | 1 | moottorin tilatieto (1 = käynnistetty, 0 = ei-käynnistetty) |
| suuntatieto | DI | Bool | 0 | 1 | tilatieto liikkeen suunnasta (1 = eteen, 0 taakse) |
| vikaantumistieto | DI | Bool | 0 | 1 | vikaantumisen tilatieto (1 = ei häiriöitä, 0 = vähintään yksi häiriö) |
| käyntitieto | DO | Bool | 0 | 1 | moottorin tilatieto (1 = käynnistetty, 0 = ei-käynnistetty) |
| suuntatieto | DO | Bool | 0 | 1 | tilatieto liikkeen suunnasta (1 = eteen, 0 taakse) |
| vikaantumistieto | DO | Bool | 0 | 1 | vikaantumisen tilatieto (1 = OK, 0 = vähintään yksi häiriö) |
| | | | | | |
| hätäpysäytys | DI | Bool | 0 | 1 | hätäpysäytyksen vaijerikytkimen tilatieto (1 = ei-aktiivinen, 0 aktiivinen) |
| käynnistä | DO | Bool | 0 | 1 | käynnistää tai pysäyttää moottorin (1 = päälle, 0 = pois) |
| suuntaa | DO | Bool | 0 | 1 | asettaa liikkeen suunnan (1 = eteen, 0 = taakse) |

***Kuva B.2*** *Rajapintamäärittely.*

# LIITE C: KENTTÄLAITTEIDEN PARAMETRIT

**Anturien parametrit**

| Valmistajan nimi | | Laitteen nimi | | Laitteen ID | | FW/SW-versio | | Käyttöönotettu | |
|---|---|---|---|---|---|---|---|---|---|
| *Valmistaja A* | | *Pyörintävahti 1* | | xx03-SS | | *A.1* | | *DD.MM.YYYY HH:MM* | |
| **Parametrin tagi** | **Parametrin nimi** | **Kuvaus** | **Raja-arvon tyyppi** | **Minimiarvo** | **Maksimiarvo** | **Suunniteltu arvo** | **Käyttöönotettu arvo** | **Mittayksikön nimi** | **Mittayksikön tunnus** |
| freqLowLim | Minimipyörimisnopeus | Ennaltamäärätty taajuden raja-arvo | Vakio | 5 | 300 | N/A | 10 | Impulssia minuutissa | Imp/min |
| freqStartDelay | Käynnistysviive | Rajakytkimen hälytys sivuutetaan | Vakio | 0 | 60 | 15 | 5 | Sekunti | s |
| | | | | | | | | | |
| **Valmistajan nimi** | | **Laitteen nimi** | | **Laitteen ID** | | **FW/SW-versio** | | **Käyttöönotettu** | |
| *Valmistaja A* | | *Pyörintävahti 2* | | xx04-SS | | *A.1* | | *DD.MM.YYYY HH:MM* | |
| **Parametrin tagi** | **Parametrin nimi** | **Kuvaus** | **Raja-arvon tyyppi** | **Minimiarvo** | **Maksimiarvo** | **Suunniteltu arvo** | **Käyttöönotettu arvo** | **Mittayksikön nimi** | **Mittayksikön tunnus** |
| freqLowLim | Minimipyörimisnopeus | Ennaltamäärätty taajuden raja-arvo | Vakio | 5 | 300 | N/A | 10 | Impulssia minuutissa | Imp/min |
| freqStartDelay | Käynnistysviive | Rajakytkimen hälytys sivuutetaan | Vakio | 0 | 60 | 15 | 5 | Sekunti | s |
| | | | | | | | | | |
| **Valmistajan nimi** | | **Laitteen nimi** | | **Laitteen ID** | | **FW/SW-versio** | | **Käyttöönotettu** | |
| *Valmistaja B* | | *Tukosvahti* | | xx03-LS | | *N/A* | | *DD.MM.YYYY HH:MM* | |
| **Parametrin tagi** | **Parametrin nimi** | **Kuvaus** | **Raja-arvon tyyppi** | **Minimiarvo** | **Maksimiarvo** | **Suunniteltu arvo** | **Käyttöönotettu arvo** | **Mittayksikön nimi** | **Mittayksikön tunnus** |
| BlockSens | Reagointiherkkyys | Kalibrointiruuvin asento | Vakio | 0 | 100 | 50 | 50 | Prosentti | % |
| BlockStartDelay | Käynnistysviive | Rajakytkimen hälytys sivuutetaan | Vakio | 0 | 60 | 2 | 2 | Sekunti | s |
| | | | | | | | | | |
| **Valmistajan nimi** | | **Laitteen nimi** | | **Laitteen ID** | | **FW/SW-versio** | | **Käyttöönotettu** | |
| *Valmistaja C* | | *Virtamittari* | | xx03-IE | | *N/A* | | *DD.MM.YYYY HH:MM* | |
| **Parametrin tagi** | **Parametrin nimi** | **Kuvaus** | **Raja-arvon tyyppi** | **Minimiarvo** | **Maksimiarvo** | **Suunniteltu arvo** | **Käyttöönotettu arvo** | **Mittayksikön nimi** | **Mittayksikön tunnus** |
| CurUpLim | Maksimivirta | Ennaltamäärätty sähkövirran raja-arvo | Vakio | 0 | 30 | 25.1 | 25 | Ampeeri | A |

***Kuva C.1*** *Poistokuljettimen antureiden parametrien määrittely.*

**Taajuusmuuttajan parametrit**

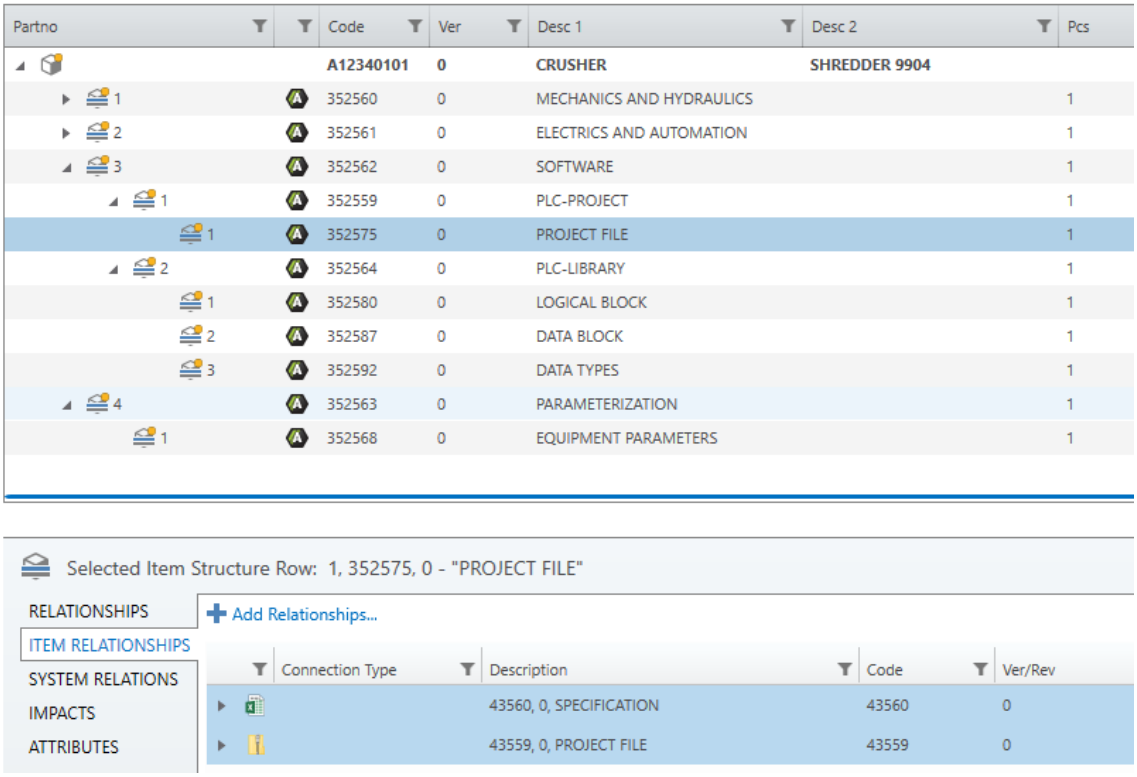
| Valmistajan nimi | | Laitteen nimi | | Laitteen ID | | FW/SW-versio | | Käyttöönotettu | |
|---|---|---|---|---|---|---|---|---|---|
| *Valmistaja D* | | *DTC-taajuusmuuttaja* | | FC1 | | *A.1* | | *DD.MM.YYYY HH:MM* | |
| **Parametrin tagi** | **Parametrin nimi** | **Kuvaus** | **Raja-arvon tyyppi** | **Minimiarvo** | **Maksimiarvo** | **Suunniteltu arvo** | **Käyttöönotettu arvo** | **Mittayksikön nimi** | **Mittayksikön tunnus** |
| Uout | Ohjausjännite | taajuusmuuttajan lähtöjännite | Dynaaminen | -400 | 400 | 0 | 0 | Voltti | V |
| UpulseMax | Maksimijännite | taajuusmuuttajan maksimipulssijännite | Vakio | -1350 | 1350 | 1350 | 1350 | Voltti | V |
| UpulseMin | Minimijännite | taajuusmuuttajan minimipulssijännite | Vakio | -1350 | 1350 | -1350 | -1350 | Voltti | V |
| Iout | Virtaviesti | taajuusmuuttajan lähtövirta | Dynaaminen | 4 | 20 | 0 | 0 | Ampeeri | A |
| Iuplim | VirranYläraja | ylikuormitusvirta | Vakio | 0 | 50 | 25 | 25 | Ampeeri | A |
| VelMax | Maksiminopeus | moottorin mekaaninen maksiminopeus | Vakio | -3600 | 3600 | 3600 | 3600 | Kierrosta per minuutti | RPM |
| VelMin | Miniminopeus | moottorin mekaaninen miniminopeus | Vakio | -3600 | 3600 | -3600 | -3600 | Kierrosta per minuutti | RPM |
| Pgain | Vahvistus | PID-algoritmin P-osan viritysparametri | Valittavissa | 0 | 100 | 25 | 27 | - | N/A |
| Itime | Integrointiaika | PID-algoritmin I-osan viritysparametri | Valittavissa | 0 | 3600 | 50 | 45 | Sekunti | s |
| Dtime | Derivointiaika | PID-algoritmin D-osan viritysparametri 1 | Valittavissa | 0 | 10 | 0 | ei-käytössä | Sekunti | s |
| D_filt_time | Derivointisuodatusaika | PID-algoritmin D-osan viritysparametri 2 | Valittavissa | 0 | 10 | 0 | ei-käytössä | Sekunti | s |
| VelSP1 | NopeudenAsetusarvo1 | ennaltamääritetty nopeus 1 | Vakio | -3600 | 3600 | 1466 | 1466 | Kierrosta per minuutti | RPM |
| VelSP2 | NopeudenAsetusarvo2 | ennaltamääritetty nopeus 2 | Vakio | -3600 | 3600 | -1466 | -1466 | Kierrosta per minuutti | RPM |

***Kuva C.2*** *Poistokuljettimen taajuusmuuttajan parametrien määrittely.*

| Raja-arvon tyyppi | Kuvaus |
|---|---|
| Vakio | Arvot fiksataan käyttöönotossa ja niitä ei saa muuttaa käyttöönoton jälkeen |
| Valittavissa | Arvoja mahdollista valita ja vaihtaa käytön aikana, noudattaen fiksattua arvoväliä |
| Dynaaminen | Arvot mukautuvat ajon aikana (laskennallisin menetelmin) |

***Kuva C.3*** *Raja-arvotyyppien kuvaus.*

# LIITE D: VERSIONHALLINNAN TUOTERAKENNE




***Kuva D*** *Versionhallinnan tuoterakenteen demonstraatio Aton-työkalulla.*

# LIITE E: KUSTANNUSARVIO JA INVESTOINTILASKELMA

| Ohjelmistokehityskustannukset nykykäytännöillä | | |
|---|---|---|
| Ohjelmistoprojekti | **Suppea** | **Laaja** |
| Kehitysaika (kk) | **3** | **6** |
| Työaika (d/kk) | 20 | 20 |
| Työpaivän pituus (h) | 7.5 | 7.5 |
| Työaika (h) | 450 | 900 |
| Työn tuntihinta (eur) | 100 | 100 |
| Kehityskustannukset yhteensä (eur) | 45000 | 90000 |
| Kustannukset (kEur) | **45** | **90** |

**Kuva E.1** *Karkea arvio projektille kehitettävälle ohjelmistolle (haarukointi).*

| Ohjelmistokehityskustannukset FAT-testauksella | |
|---|---|
| Tuotantolinjan ohjelmisto | |
| **Ohjelmistokehityksen vaihe** | **Aika (h)** |
| Vaatimusten määrittely ja suunnittelu | 1 |
| Ohjelmointi ja sisäinen testaus | 2 |
| Tehdashyväksyntätesti (FAT) | 0.5 |
| I/O-pisteet (kpl) | **147** |
| Työaika yhteensä (h) | 514.5 |
| Työaika yhteensä (d) | 68.6 |
| Työn tuntihinta (eur) | 100 |
| Kehityskustannukset yhteensä (eur) | 51450 |
| Kustannukset (kEur) | **51** |

**Kuva E.2** *Arvio yksittäiselle tuotantolinjalle I/O-pisteiden lukumäärän mukaan.*

| Ohjelmistokehityskustannukset FAT-testauksella | |
|---|---|
| Murskaimen ohjelmisto | |
| **Ohjelmistokehityksen vaihe** | **Aika (h)** |
| Vaatimusten määrittely ja suunnittelu | 1 |
| Ohjelmointi ja sisäinen testaus | 2 |
| Tehdashyväksyntätesti (FAT) | 0.5 |
| I/O-pisteet (kpl) | **97** |
| Työaika yhteensä (h) | 339.5 |
| Työaika yhteensä (d) | 45.3 |
| Työn tuntihinta (eur) | 100 |
| Kehityskustannukset yhteensä (eur) | 33950 |
| Kustannukset (kEur) | **34** |

**Kuva E.3** *Arvio yksittäiselle murskaimelle I/O-pisteiden lukumäärän mukaan.*

| Investointilaskelma (ohjelmistotuoteperheille) | | | |
|---|---|---|---|
| Alkuinvestointikerroin | | **3.0** | [2.0 ... 3.0] |
| Työmääräkerroin | | **0.5** | [0.1 ... 0.5] |
| Toimitukset (vuonna 2016) | SRF-tuotantolinjat | 6 | kpl |
| | Jätemurskaimet | 7 | kpl |
| Kehitys-kustannukset | Tuotantolinjan ohjelmisto | 51 | kEur |
| | Murskaimen ohjelmisto | 34 | kEur |
| Vuotuiset tuotot | Tuotantolinjan tuoteperheet | 153 | kEur |
| | Murskaimen tuoteperhe | 119 | kEur |
| Investoinnin kustannukset | Tuotantolinjan moduulit | 153 | kEur |
| | Murskaimen moduulit | 102 | kEur |
| Takaisinmaksu-aika | Tuotantolinjan tuoteperhe | **1.0** | vuotta |
| | Murskaimen tuoteperhe | **0.9** | vuotta |

***Kuva E.4*** *Modulaaristen ohjelmistotuoteperheiden takaisinmaksuaika.*

| Työkaluja mallipohjaiseen ohjelmistokehitykseen | |
|---|---|
| *Työkalusetti (vaihtoehto 1)* | *Hinta (kEur)* |
| Simulink PLC Coder | 9.1 |
| Stateflow | 2.9 |
| Kustannukset (kEur) | **12** |
| | |
| *Työkalusetti (vaihtoehto 2)* | *Hinta (kEur)* |
| SIMATIC Target 1500S for Simulink | 1.4 |
| MATLAB Coder | 5.8 |
| Simulink Coder | 2.9 |
| Stateflow | 2.9 |
| Kustannukset (kEur) | **13** |
| | |
| **Työpajojen minimihinta (2 konsulttia)** | |
| Työpajan kesto (h) | 6 |
| Työpajan hinta (kEur) | 2.2 |
| Työpajojen lukumäärä (kpl) | 2 |
| Kustannukset (kEur) | **4** |

***Kuva E.5*** *Mahdolliset lisäinvestoinnit.*

# LIITE F: MALLIPOHJAISEN SUUNNITTELUN TYÖNKULKU

Prosessilaitteen mallipohjaisen suunnittelun työnkulku ja siihen soveltuvat suunnittelutyökalut voidaan esittää seuraavasti:

- Suunnitteluvaatimusten määrittely ja hallinta
  - Asiakasvaatimusten kerääminen ja jäsentely taulukkomuotoon (MS Excel)
  - Toiminnallisten vaatimusten kuvaaminen käyttötapauskaaviolla (MS Visio)
  - Yksityiskohtien lisääminen vaatimuksiin tekstinkäsittelyohjelmalla (MS Word)
  - Vaatimusten linkittäminen toisiinsa (Aton)
- Toiminnallisen mallin luominen eli modulaarisen rakenteen kuvaus
  - Systeemin perustoiminnallisuuksien kuvaaminen luokkakaaviolla (MS Visio)
  - Hierarkkisen systeemimallin modulaarisen rakenteen luominen (Simulink)
  - Vaihtoehtoisten toiminnallisten suunnittelumallien ylläpitäminen (Aton)
- Loogisen mallin luominen eli dynaamisen käyttäytymisen kuvaus (Simulink)
  - Systeemin käyttäytymismallien määrittely
  - Käyttäytymismallien liittäminen systeemin moduuleihin
- Jäljitettävyyslinkkien luominen toiminnallisen ja loogisen mallin välille (Aton)
- Mekaniikan konseptimallin luominen
  - Karkean 3D CAD-mallin määrittely (SolidWorks)
  - Kinematiikan ja geometrian määrittely (Simscape Multibody)
  - Mekaanisten komponenttien liittäminen loogiseen malliin (Simulink)
- Abstraktien toimilaitteiden lisääminen (Simulink)
  - Rajapintamäärittelyjen mukaisten nopeusrajoitettujen toimilaitteiden lisääminen
  - Rajapintamäärittelyjen mukaisten asentorajoitettujen toimilaitteiden lisääminen
- Aikapohjaisten toimintojen määrittely (Stateflow)
  - Toimilaitteiden ohjaustapojen määrittely
  - Ohjaussekvenssien ajoituksien määrittely (esim. viiveet ja näytteistysväli)
  - Aikapohjaisten toimintojen liittäminen loogiseen malliin
- Abstraktien antureiden lisääminen (Simulink)
  - Rajapintamäärittelyjen mukaisten, rajakytkiminä toimivien, antureiden lisääminen
- Tapahtumapohjaisten toimintojen määrittely (Stateflow)
  - Antureiden toimintojen määrittely
  - Toimilaitteiden ohjaussignaalien määrittely
  - Tapahtumapohjaisten toimintojen liittäminen loogiseen malliin
- Mekaniikan konseptimallin korvaaminen yksityiskohtaisella mekaniikkamallilla

- Yksityiskohtaisten geometria- ja kinematiikkatietojen tuonti (Simscape Multibody)
- Karkeiden mekaniikkakomponenttien korvaaminen yksityiskohtaisilla (Simulink)

- Yksityiskohtaisten antureiden ja toimilaitteiden valitseminen
  - Antureiden valitseminen kirjastosta (Simscape)
  - Toimilaitteiden valitseminen kirjastosta (Simscape Electrical ja Simscape Fluids)
- Automaatiosovelluksen kehitys (Simulink PLC Coder)
  - Automaattisesti generoidun lähdekoodin tuominen PLC-työkalulle (TIA Portal)
  - Kehitetyn ohjelmistomoduulin lähettäminen logiikkaohjaimeen (Step 7)
- Testitapausten määrittely ja luominen (käyttötapauskaavion pohjalta)
  - Realististen anturi- ja toimilaitesignaalien määrittely (MS Excel)
  - Simulointiparametrien hienosäätö (MATLAB)
  - Testitapausten herätesignaalien luominen (Simulink)
- PLC-ohjelmiston HIL-testaus ja validointi (Simulink)
- Ohjelmiston ja simulointitietojen vieminen versionhallintaan (Aton)

# LIITE G: OHJELMISTOTUOTANNON KEHITYSKOHTEET TIIVISTETTYNÄ

Ohjelmistojen tuottamiseen liittyvät kehityskohteet voidaan tiivistää seuraavasti:

- Resurssien lisääminen (ohjelmisto-osaaminen organisaation sisällä)
- Valvonnan lisääminen (kehitettävien ohjelmistojen valmiusasteen seuranta)
- Aikataulun laadinta (ohjelmistokehityksen suunnittelu ja aikataulutus)
- Testaus (ohjelmistojen mallipohjainen testaus ja FAT-menettely)
- Dokumentaatio (tarkemmat toiminnankuvaukset tuotekehittäjien laatimana)
- Versionhallinta (Käyttöönotto ja moduulikirjastojen laatiminen)
- Kantaversiot (vaatimusten, toiminnallisuuksien ja implementointien määrittely)